TREBALL FINAL DE GRAU

TÍTOL DEL TFG: Disseny d'un algorisme d'optimització de preus en viatges multi-destí

TITULACIÓ: Grau en Enginyeria d'Aeronavegació

AUTOR: Yanik Lacroix Torrent

DIRECTOR: Susana Clara López Masip

DATA: 10 de juliol del 2015

Títol: Disseny d'un algorisme d'optimització de preus en viatges multi-destí

Autor: Yanik Lacroix Torrent

Director: Susana Clara López Masip

Data: 10 de juliol del 2015

Resum

En els darrers anys el món de l'aviació, impulsat per la innovació tecnològica, ha crescut considerablement. En paral·lel hi ha hagut un gran creixement en quant al nombre d'aeroports, trajectes entre aeroports i companyies aèries que els operen. Un dels efectes d'aquest gran creixement és l'àmplia oferta de bitllets d'avió, fins i tot, per un mateix trajecte. En els últims anys el paper de les agencies de viatge convencionals, en la venda de vols, ha estat substituït en gran part pels cercadors de vols coincidint amb la generalització de l'ús d'internet.

Aquest projecte s'integra en l'àmbit de la recerca de bitllets d'avió a través de la xarxa i s'adreça especialment a les persones decidides a viatjar i visitar diferents ciutats, però que a l'hora són flexibles tant en les destinacions com, en cert grau, a les dates. Proposa un model de cercador innovador i més flexible que els cercadors existents, tant pel que fa les dates escollides, en lloc de fixar dies concrets es fixa una franja de dies per viatjar, com les ciutats, que no es fixen a priori.

Per dur a terme aquest avenç s'ha dissenyat un model basat en digrafs amb pesos, sobre els que s'aplica un algorisme bàsic de teoria de grafs que s'anomena *Dijkstra*. El digraf modela la xarxa d'aeroports dels quals es disposen dades (en aquest cas aeroports europeus) i els pesos són els preus dels trajectes entre aeroports, obtinguts a partir d'una base de dades. En aquest model l'usuari fixa una ciutat d'origen, una franja de possibles dies de viatge i una durada màxima/mínima del viatge, el resultat de la recerca és un llistat d'opcions de viatge. En cada opció apareix el preu, les diferents ciutats que es visiten i els dies de vol.

El fet que l'usuari no fixa els destins a priori és innovador i permet obtenir resultats d'optimització alternatius als existents per viatges amb destinació múltiple.

Title: An optimization algorithm design for multi-destination trips

Author: Yanik Lacroix Torrent

Director: Susana Clara López Masip

Date: July 10 th 2015

Overview

Thanks to the technological innovation, the world of aviation has grown considerably in the last years. Concerning the number of airports and the airlines there has been a considerable growth too. One of the effects of this growth is the rise of the offer of flight tickets, even in the same journey. In the last years, the role of the travel agency in the sale of tickets has been replaced by online flight search engines, taking advantage of the widespread use of internet.

This project is included in the field of flights search through the network and is aimed at those people who are decided to travel and who want to visit many cities, but who also are flexible concerning the destination and the dates.
The project proposes an innovative model of an online searcher that is more flexible than the conventional search engines because of the ability to choose a period of days to travel and its originality concerning the cities that are not fixed *a priori.*

To achieve this goal we have designed a model based on weighted digraphs, in which we have applied a basic graph theory algorithm called *Dijkstra*. In this case, the digraph represents a network of European airports with data available and the weights represent the different travel prices between the airports, obtained through a data base. In this model, the user fixes a city of origin, a period of travelling days and the duration of the trip. Thus, the search result is a list of travel options. The travel price, the cities and the travel period appear in every option.

The fact that the user can't set the destination is an innovative option and provides alternative optimized results in contrast to the existing multi-destination trips.

ÍNDEX

INTRODUCCIÓ

En els darrers anys el món de l'aviació, impulsat per la innovació tecnològica, ha crescut considerablement. En paral·lel hi ha hagut un gran creixement en quant al nombre d'aeroports, trajectes entre aeroports i companyies aèries que els operen. Un dels efectes d'aquest gran creixement és l'àmplia oferta de bitllets d'avió fins i tot per un mateix trajecte. En els últims anys el paper de les agencies de viatge convencionals, en la venda de vols, ha estat substituït en gran part pels cercadors de vols coincidint amb la generalització de l'ús d'internet. Un dels sectors que més ha aprofitat aquest fet ha estat el de la venta de bitllets d'avió *online*. En aquest àmbit, han aparegut un gran ventall de possibilitats en relació a la recerca i obtenció de bitllets d'avió econòmics. Els cercadors de vols són una eina cada cop més utilitzada per la població, ja que permeten obtenir bitllets econòmics sense necessitat d'intermediaris. Actualment és possible trobar una gran diversitat de cercadors, i aquesta diversitat s'acompanya d'una població exigent, que busca obtenir bitllets d'avió al menor preu possible i en dates concretes.

Aquest projecte s'integra en l'àmbit de la recerca de bitllets d'avió a través de la xarxa i s'adreça especialment a les persones decidides a viatjar i visitar diferents ciutats, però que a l'hora són flexibles tant en les destinacions com, en cert grau, a les dates. L'objectiu d'aquest projecte és dissenyar un model de buscador que permeti fer una recerca d'itineraris per diferents ciutats, com a mínim dues, proporcionant les opcions més econòmiques del mercat. Aquest model de buscador ha de gestionar una recerca alternativa a la que ja existeix en el mercat, oferint a l'usuari molta més flexibilitat a l'hora de buscar vols, especialment pel que fa a les dates i les ciutats, tot prioritzant el preu del producte.

Per assolir l'objectiu plantejat, és necessària una base matemàtica per tal de dissenyar un model de recerca basat en l'optimització de preus. El model que es proposa utilitza en digrafs amb pesos, sobre els que s'aplica un algorisme bàsic de teoria de grafs que s'anomena *Dijkstra* i que permet trobar el camí més curt entre dos punts. El digraf modela la xarxa d'aeroports dels quals es disposen dades (en aquest cas aeroports europeus) i els pesos són els preus dels trajectes entre aeroports, obtinguts a partir d'una base de dades. En aquest model l'usuari fixa una ciutat d'origen, una franja de possibles dies de viatge i una durada màxima/mínima del viatge, el resultat de la recerca és un llistat d'opcions de viatge. En cada opció apareix el preu, les diferents ciutats que es visiten i els dies de vol.

El mètode de programació que s'ha fet servir al llarg del projecte és el *matlab,* que permet executar el model i obtenir resultats de manera eficient. La idea del projecte és que el programa creat pugui implementar-se dins de l'entorn de la pàgina web, havent comprovat prèviament que l'algorisme funciona correctament.

Abans d'entrar en detall convé presentar alguns dels cercadors disponibles a la xarxa i les seves utilitats més destacades. Així mateix cal introduir el concepte de base de dades i quina és la font d'informació dels cercadors. En el Capítol 1 de la memòria s'expliquen les característiques principals d' alguns dels cercadors més significatius per entendre l'aportació del projecte. Per la seva singularitat i proximitat amb el projecte s'ha dedicat el Capítol 2 a altres iniciatives de recerca de vols més innovadores. Les bases matemàtiques, en particular la descripció dels digrafs amb pesos i la descripció de l'algorisme *Dijkstra* s'introdueixen en el Capítol 3. La construcció del model es basa en la definició d'una matriu en pesos, sobre la que posteriorment s'aplicarà l'algorisme de *Dijkstra*. La descripció d'aquesta matriu és el contingut principal del Capítol 4. El *software* de programació que s'ha utilitzat ha estat el *matlab.* S'han creat una sèrie de funcions, unes bàsiques i unes altres més complexes per automatitzar tot el procés de modelització del problema. Aquesta part d'informàtica és el contingut del Capítol 5. La memòria acaba amb l'exposició dels resultats obtinguts, Capítol 6, i la presentació de les conclusions, Capítol 7.

CAPÍTOL 1. Cercadors Existents

1.1 Cercadors

En el context de la compra de vols, existeixen diferents tipus de cercadors. Internet és una font d'aquests motors de recerca que faciliten la troballa de bitllets d'avió econòmics que han esdevingut un sistema àmpliament utilitzat per la població. En els darrers anys el nombre de pàgines webs dedicades a la recerca de vols s' ha incrementat de manera molt ràpida degut al creixement del sector de l'aviació i a la generalització de l'ús d'internet. Actualment existeix un ampli ventall de pàgines web per cercar ràpidament un bitllet d'avió. Tot i així, degut a l'abundància de cercadors, és difícil saber quin és el mètode més eficaç i el que millor s'adapta a la nostra demanda.

Aquest projecte pretén cobrir un espai en aquest sector . S'adreça especialment a les persones decidides a viatjar i visitar diferents ciutats, però que a l'hora són flexibles tant en les destinacions com, en cert grau, a les dates. Abans d'entrar en detall convé presentar alguns dels cercadors disponibles a la xarxa i les seves utilitats més destacades. Així mateix cal introduir el concepte de base de dades i quina és la font d'informació dels cercadors.

A continuació s'expliquen les característiques principals d' alguns dels cercadors més significatius pel desenvolupament d'aquest projecte.

1.1.1 eDreams

eDreams [9] és un cercador que ofereix una gran varietat de productes, tant comparació de vols com hotels o paquets de vacances. Permet fer la compra a través de la seva pàgina web http://www.edreams.es.

Figura 1: Cercador eDreams.

En el cas dels bitllets d'avió, aquesta pàgina web no només té la funció de cercar sinó que també aprofita l'oportunitat de mercat per a fer d'intermediari entre el client i la companyia aèria i així poder cobrar tant comissions com despeses de gestió entre d'altres. Aquestes pàgines web s'anomenen OTA (Online Travel Agency), i es caracteritzen per fer negoci treballant com a intermediaris.

Per tal de trobar un bitllet econòmic, les opcions que s'ofereixen són: buscar un únic bitllet d'anada, d'anada i tornada o bé destinacions múltiples. En els tres casos, els paràmetres que hem d'entrar són la data dels trajectes i les ciutats d'origen i destinació. A partir d'aquí es pot restringir la recerca proposant una hora de sortida, triant la classe de cabina en la que volem viatjar o especificant si volem vols directes o estem disposats a fer escala. (Vacaciones eDreams, 1999)

1.1.2 Kayak

Kayak [3] també és un gran cercador que no només ofereix als seus clients buscar vols econòmics, sinó que també, com fa eDreams, ofereix molts altres productes com ara hotels, cotxes o paquets de viatges. La seva pàgina web és http://www.kayak.es/flights

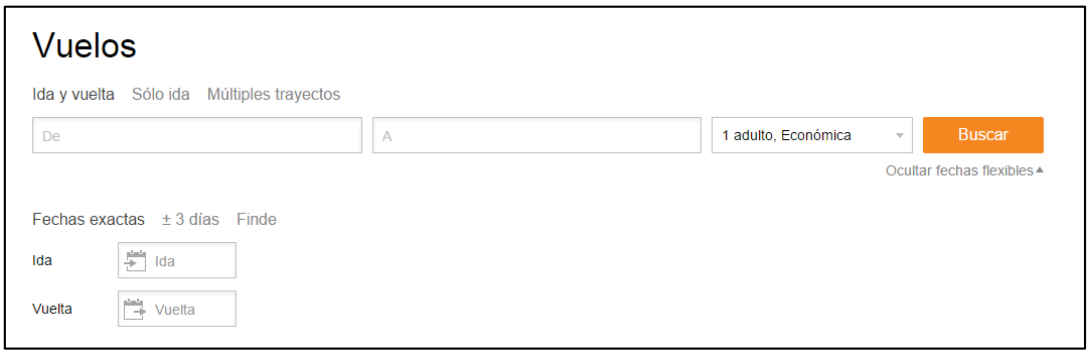


Figura 2: Cercador Kayak.

Pel que fa les opcions de recerca trobem les mateixes opcions que amb eDreams: es poden buscar bitllets d' anada, d'anada i tornada o múltiples destins. A partir d'aquí, l'usuari ha d'introduir les dates i les ciutats d'origen i destí. Kayak té una mica més de flexibilitat que eDreams, ja que es poden seleccionar les dates amb 3 dies de marge, la qual cosa dóna un ventall més ampli de solucions. També dóna més flexibilitat pel que fa a l'origen i al destí, ja que permet incloure els aeroports propers als que estem buscant. Així doncs incrementa també el ventall de solucions ja que inclou vols que fan trajectes molt similars al sol·licitat inicialment.

A diferencia d'eDreams, aquesta web redirecciona els seus clients a la OTA més econòmica i no a la companyia aèria. Aquesta pàgina web rep comissions pel

nombre d'usuaris que redirecciona, però no per despeses de gestió com en el cas de les OTA.

1.1.3 SkyScanner

Skyscanner [8] és un cercador dedicat principalment a la cerca de vols econòmics, tot i que també es poden trobar les opcions de recerca d'hotel i lloguer de cotxes, però aquestes funcions no estan tant desenvolupades com la que a nosaltres ens interessa, la dels vols.

Les opcions de recerca són semblants als casos anteriors. Els paràmetres d'entrada que ens demana són les dates, que en aquest cas són encara més flexibles que en els dos anteriors, ja que es pot escollir si volem visualitzar els preus inclosos en un únic dia, un setmana o bé un mes sencer. A més es pot veure en una gràfica de barres els preus més econòmics dins del període escollit. Hem de seleccionar també la ciutat d'arribada i la de sortida, però amb la novetat següent: a més de poder seleccionar els aeroports propers es poden seleccionar també les sortides o arribades des d'un país en concret, i cercar el camí més barat a qualsevol aeroport d'aquest país. Són dues característiques molt innovadores que han proporcionat molta més llibertat a l'hora de buscar.

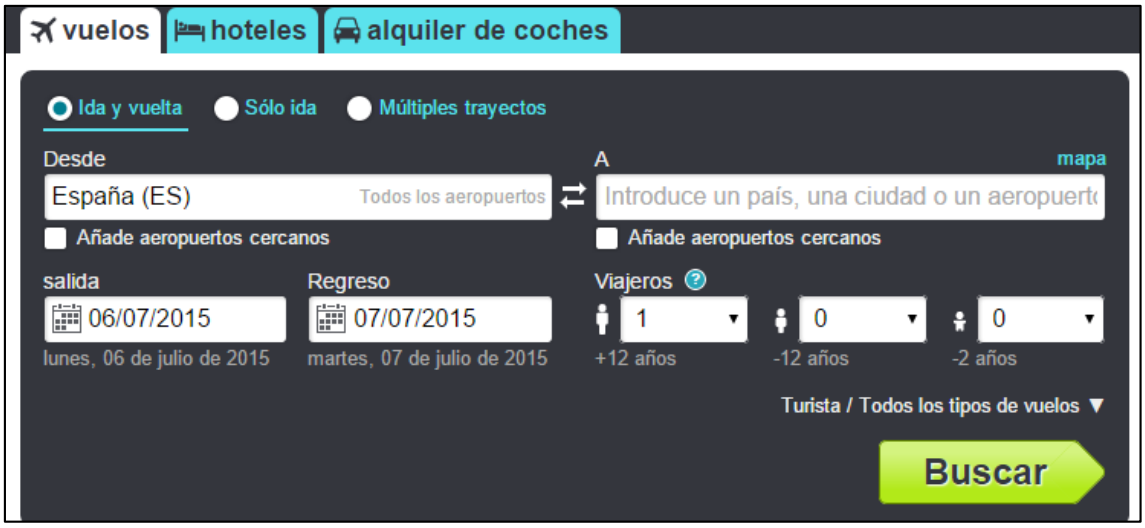


Figura 3: Cercador SkyScanner.

Una de les eines que també ofereix Skyscanner, molt interessant, és la seva opció de mapa, on es poden veure, per exemple, totes les destinacions, que ens ofereix una ciutat en concret. Skyscanner també redirecciona els seus clients a la OTA o directament a la companyia aèria més econòmica.

1.1.4 Google Flights

Google Flights [1] és una plataforma dedicada exclusivament a la recerca de bitllets d'avió. No ofereix altres productes com la majoria de pàgines que ja hem vist. Es poden triar viatges d'anada, anada i tornada i també múltiples destins. Cal entrar un interval de dates, que en aquest cas no és variable, un o més aeroports d'origen i destí. És un cercador que dóna molta llibertat i té moltes opcions afegides que poden ser molt útils. Destaca perquè deixa triar tots els aeroports que s'hi vulguin incloure. També s'hi poden afegir paràmetres opcionals per a restringir més la recerca, com ara la classe en la que volem viatjar (*economy*, *business, first class*...), el número d'escales que es volen fer com a màxim, la companyia o aliança amb la que es vol viatjar, la duració del vol o el preu.

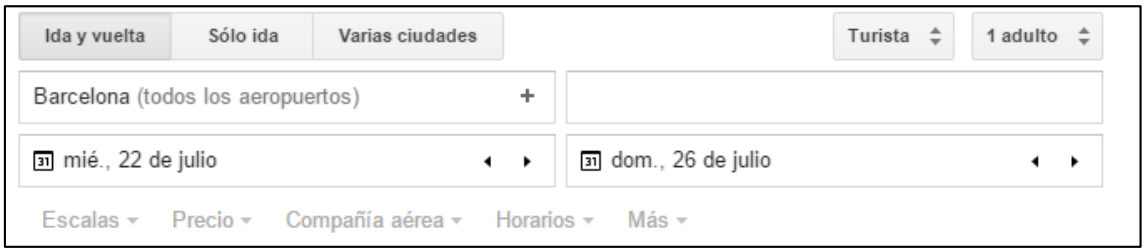


Figura 4: Cercador GoogleFlights.

Google Flights és una pàgina web que redirecciona directament els seus clients a la pàgina web de la companyia aèria, per tant tampoc es cobren despeses de gestió.

1.2 Taula de Comparacions

A continuació es presenta una taula amb les característiques resumides de tots els cercadors explicats prèviament i que tenen un cert interès per poder començar a parlar de les prestacions que oferirà el nostre motor de recerca.

Buscador	Ciutat Origen	Ciutat Destí	Data Anada	Data Tronada	Flexibilitat Dates	Grans Grups Ciutats	Aeroports propers
eDreams	Y	Y	Y	Y	N	N	N
Kayak	Y	Y	Y	Y	Y	N	Y
SkyScanner	Y	Y	Y	Y	Y	Y	Y
Google Flights	Y	Y	Y	Y	N	Y	Y

Taula 1 : Comparació dels principals cercadors.

(Posem Y per indicar que el cercador permet seleccionar l'ítem corresponent i N si no ho permet)

1.3 Dades dels Cercadors

En aquesta secció s'explica què hi ha darrera d'un cercador de vols, és a dir, quina és la informació de la que disposen els cercadors i que els hi permet, un cop sol·licitada una determinada recerca, oferir la resposta a l'usuari per tal d'intentar satisfer totes les seves necessitats. Els cercadors poden buscar directament la informació a les pàgines web de les companyies aèries, una tasca no gens fàcil degut als diferents *scripts* que hi ha darrera de cada pàgina i que cada una té el seu propi format. L'altra manera que tenen els cercadors o meta cercadors de dur a terme aquesta tasca és buscar la informació en altres pàgines webs que no són les de les companyies aèries però que també mostren preus. Aquestes pàgines s'anomenen cercadors de cercadors.

Amb la informació recopilada de totes les recerques que es fan mitjançant els cercadors, es poden acumular grans volums de dades, que s'organitzen en bases de dades. Es poden trobar diferents tipus de base de dades en el mercat de l'aviació, que es troba en constant creixement i canvi, degut al seu ràpid desenvolupament. Hi ha bases de dades que accedeixen directament a la informació de les companyies i que habitualment s'actualitzen cada dia, però la gran majoria són de pagament i d'accés restringit. Hi ha empreses informàtiques que extreuen de les pàgines web de totes les companyies aèries del món els preus actualitzats per poder-los vendre o fer-ne estudis molt més exhaustius. Una d'aquestes empreses és QL2 [6], que rastreja les pàgines web de les companyies aèries, creuers i molts altres camps per fer negoci amb tota aquesta informació. També es poden trobar altres tipus de bases de dades, com ara la que nosaltres farem servir, i que estan basades en les recerques dels diferents usuaris. Els resultats de les recerques es guarden i l'accés a aquesta informació és més senzill.

Figura 5: Logo de l'empresa QL2.

CAPÍTOL 2. Emprenedors

En els darrers anys han sorgit moltes idees per intentar fer possible viatges d'avió a un preu més econòmic, sobretot de cara als joves. Hi ha hagut moltes idees innovadores per cercar la manera d'economitzar al màxim a l'hora de viatjar en aquest mitjà de transport. Històricament l'avió ha estat un transport utilitzat bàsicament per empresaris o bé per les famílies en el context de vacances, ja que era i segueix sent un mitjà de transport molt car. L'aparició de les companyies *low cost* ha facilitat l'accés d'aquest últim a viatgers joves i a la població de classe mitjana. Tot i així, davant l'ampli ventall de possibilitats que es troben, és necessari algun tipus d'eina per tal de poder triar la millor opció.
Tot seguit es presenten alguns projectes innovadors, en l'àmbit de la troballa de vols econòmics i que s'allunyen una mica del patró de cercadors de vols habituals.

2.1 WaynaBox

WaynaBox [11] és una iniciativa que permet als joves viatjar econòmicament en avió. La idea principal és aprofitar els seients buits que les companyies aèries no venen per tal de poder treure'ls-hi profit. L'empresa WaynaBox té convenis amb diferents companyies aèries que li permeten obtenir bitllets de darrera hora a un preu molt reduït. L'inconvenient d'aquest sistema és que, com les companyies volen vendre el màxim de bitllets al màxim preu possible, no es pot saber les places que hi haurà disponibles a cada vol fins a poques hores abans de la seva sortida. WaynaBox ofereix un producte per a viatgers, el destí dels quals no és el més rellevant, ja que aquest no es pot saber fins poc temps abans del viatge. Cal esmentar que el viatge només pot realitzar-se durant els caps de setmana i aquest té un preu tancat.

Figura 6: Breu descripció del funcionament de WaynaBox.

Els usuaris han de triar un cap de setmana en el que volen viatjar. Es proposen unes ciutats. Afegint un suplement, es poden descartar algunes de les ciutats proposades. D'aquesta manera, mitjançant un preu tancat i econòmic, els joves (hi ha una restricció d'edat) que vulguin poden visitar una de les ciutats

proposades durant un cap de setmana. Cal dir que és una iniciativa que està creixent i de moment està centrada en el mercat europeu.

2.2 SkipLagged

Skiplagged [7] és també una iniciativa emprenedora per a poder cercar vols de la manera més econòmica possible des d'un mètode alternatiu. En aquest cas es tracta d'uns joves americans que han creat un mètode per aprofitar els preus que les gran companyies aèries ofereixen en viatges de llarg recorregut. Han vist que alguns bitllets d'avió, fent escala, són mes econòmics que els vols directes a la ciutat de l'escala. Sobre aquesta premissa, un vol que vagi de Paris a San Francisco, per exemple, fent escala a Toronto, pot ser més barat que un vol de Paris a Toronto. Sabent això i sense portar equipatge facturat, es pot comprar el vol a San Francisco, fent escala a Toronto, que és més econòmic que el vol de Paris a Toronto, i baixar quan l'avió faci escala. D'aquesta manera fem el mateix trajecte a un preu més baix. Mitjançant aquest mètode s'arriben a descomptes de fins al 60%.

Figura 7: Cercador Skiplagged.

2.3 El Projecte

S'han estudiat moltes de les eines que hi ha actualment al mercat per poder planificar un viatge en avió i s'ha trobat que hi ha un camp que encara no està cobert: és l'àmbit dels viatges amb destinació múltiple on els usuaris estan oberts a qualsevol destinació possible. Com hem vist WaynaBox ofereix un producte molt similar, però a un preu tancat. En aquets cas, l'usuari podrà triar el preu i el destí que més li convingui, dins l'oferta que se li plantejarà, a partir d'una franja de disponibilitat de dates i un aeroport de sortida que triarà l'usuari.

Per això es proposa un cercador on és imprescindible donar una ciutat de sortida, des d'on començarà el viatge. La ciutat o ciutats de destí, en canvi, no són fixades per l'usuari, sinó que es donen en funció del preu. Així, es busquen recorreguts econòmics per a la gent que vol viatjar per Europa sent el destí una variable que l'usuari no decideix. Pot passar, però que hi hagi ciutats on l'usuari no vulgui viatjar, ja sigui perquè ja hi ha estat o perquè no li interessen. En aquest cas es permet certa flexibilitat per descartar algunes ciutats. Les dates són molt més flexibles que en tots els casos que hem vist anteriorment, essent possible la introducció de 2 períodes de dates. El primer són tots els dies que l'usuari està disponible per viatjar i el segon, el número de dies aproximat que vol estar de viatge. Per exemple: un estudiant que té tot l'estiu lliure i que pot viatjar durant els tres mesos de vacances, selecciona 3 mesos disponibles per viatjar. Això no

vol dir que estigui viatjant els tres mesos, perquè ha de seleccionar un altre període que és el número de dies aproximats que vol estar de viatge. Així podria seleccionar, per exemple, un període de 8-10 dies. Amb aquesta flexibilitat en les dates i oferint els destins més econòmics (sempre amb la possibilitat de poder-ne descartar els que es desitgin) s'ofereix una nova possibilitat en l'àmbit de la recerca de vols.

CAPÍTOL 3. Descripció Bases Matemàtiques

Per programar el cercador de vols plantejat anteriorment, s'introdueixen a continuació les bases matemàtiques necessàries per dissenyar el model utilitzat pel cercador. L'eina principal han estat els digrafs amb pesos en els arcs. S'introdueix primerament el vocabulari bàsic necessari per entendre la teoria de grafs utilitzada.

3.1 Digrafs i distàncies

Intuïtivament un graf és una xarxa d' "objectes" que estan interrelacionats d'alguna manera. Es poden trobar molts casos de grafs en el món real, un exemple és una xarxa de metro, on totes les estacions són els objectes i estan relacionades entre elles, de manera que només des d'unes estacions determinades es pot anar a unes altres amb connexió directa. El diagrama o mapa de les línies de metro és un graf, on els seus vèrtex són les estacions de metro. Les estacions de metro estan interconnectades per arestes, i l'existència d'una aresta entre dues estacions A i B de la xarxa indica que es pot anar de A a B de forma directa, sense passar per cap altra estació.

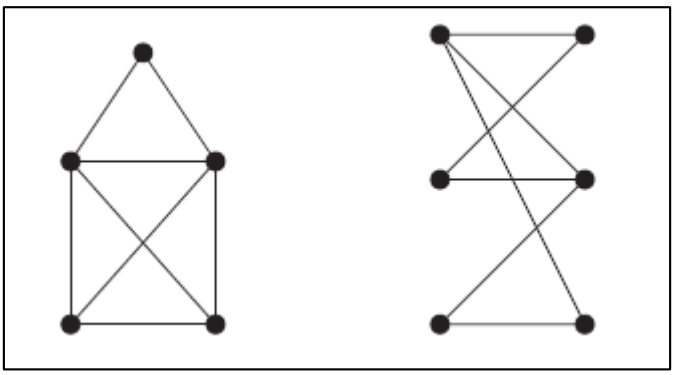


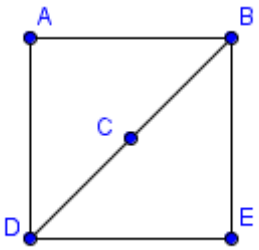
Figura 8: Exemple de graf. [5]

Formalment, un graf G és una parella de conjunts (V,E) on V representa el conjunt de vèrtexs i E el conjunt d'arestes, és a dir de parelles de vèrtexs. S'escriu uv per indicar {u, v}. Si uv és una aresta de G, es diu que u i v són vèrtexs adjacents.

En aquest projecte s'utilitzen digrafs, que són una generalització dels grafs. Un digraf D és una parella de conjunts (V,E) on V representa el conjunt de vèrtexs i E el conjunt d'arcs, és a dir de parelles ordenades de vèrtexs. Si (u, v) és un arc de D aleshores es diu que v és un successor de u i també que u és un predecessor de v.

Cada graf o digraf té una matriu d'adjacència on queden reflectides les adjacències entre els vèrtexs. Cada fila i columna correspon a un vèrtex, per tant s'obté una matriu [n x n] on n es el nombre total de vèrtex del digraf. En el cas que hi hagi una arc del vèrtex a al vèrtex b, la matriu d'adjacència en la posició

(a,b) tindria un 1 i si no n'hi ha cap té un 0. De tal manera que sense el diagrama del digraf, només tenint la matriu ja es poden saber totes les característiques del digraf. En el cas dels grafs la matriu d'adjacència és simètrica.



$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

Figura 9: Exemple de graf amb la seva matriu d'adjacència corresponent.

Es pot definir com a *camí* (dirigit) una seqüencia de vèrtexs en un (di) graf, tals que existeix un aresta (arc) entre cada vèrtex i el següent. Es pot dir que dos vèrtexs estan connectats si existeix un camí que vagi d'un a l'altre, sinó estaran desconnectats. El número d'arestes en un camí equival a la seva *longitud:* els vèrtexs adjacents estan connectats per un camí de longitud 1 i els segons amb un camí de longitud 2. Es pot mesurar la *distància entre dos vèrtexs* com la longitud mínima de tots els camins que els uneixen.

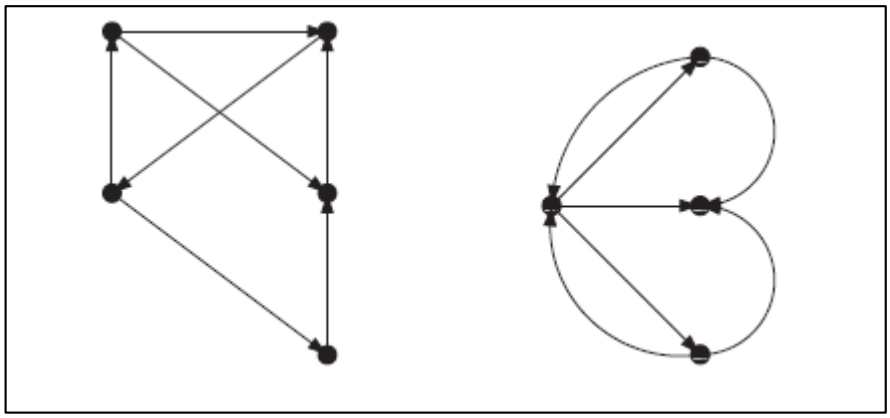
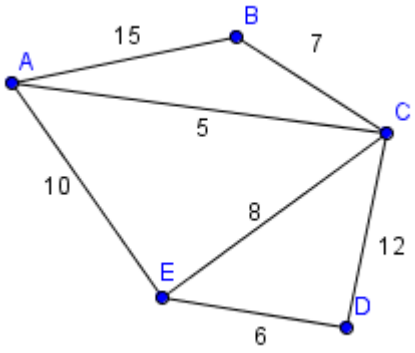


Figura 10: Exemple de graf dirigit. [5]

En aquest projecte es treballa amb *digrafs ponderats* que tenen associat un valor o *pes* a cada arc. En aquest cas es defineix la *longitud* d'un camí com la suma de tots els *pesos* o valors dels arcs. Donats dos vèrtexs u i v, la distància de u a v es defineix com la mínima longitud entre tots els camins dirigits de u a v.

Tornant al cas de les estacions de metro, el pes entre dos vèrtexs podria referir-se a la distància que hi ha entre les dues estacions.



$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 0 & 15 & 5 & 0 & 10 \\ 15 & 0 & 7 & 0 & 0 \\ 5 & 7 & 0 & 12 & 8 \\ 0 & 0 & 12 & 0 & 6 \\ 10 & 0 & 8 & 6 & 0 \end{pmatrix} \end{array}$$

Figura 11: Exemple de graf ponderat amb la seva matriu de pesos.

Les connexions entre diferents ciutats amb vols es poden representar amb un digraf amb pesos. Per exemple suposant que es té informació dels vols del 30 de gener de 2015 entre 4 ciutats (BCN, LON, PAR, OSL) amb preus segons la taula.

	BCN	LON	PAR	OSL
BCN	-	25	-	30
LON	35	-	25	-
PAR	-	40	-	35
OSL	-	35	40	-

Taula 2: Valor dels preus per anar de la ciutat X a Y.

Es pot associar la taula a un digraf amb pesos. Els vèrtexs són els aeroports dels que es disposa informació. Hi ha un arc de la ciutat A a la ciutat B si hi ha un vol directe de A a B i el pes d'aquest arc correspon al preu del vol. Es poden trobar diferents camins per anar de BCN a PAR però cap d'ells directe, i el de preu mínim és el que fa el trajecte BCN-LON-PAR. Aquets preu correspon a la distància entre BCN i PAR introduït anteriorment.

3.2 Dijkstra – Shortest Path Algorithm

3.2.1 L'algorisme

Partint d'un digraf amb pesos en els arcs es pot buscar el camí més curt, és a dir, el que té suma de pesos mínim, per arribar des d'un vèrtex fins a un altre. Per aquest propòsit es fan servir algoritmes dissenyats per obtenir el camí més curt des d'un vèrtex fins a tots els altres. Aquests algoritmes acaben proporcionant la longitud total del trajecte, i en alguns casos, tots els vèrtex pels quals haurà de passar.

L'algorisme per trobar el camí més curt que s'ha utilitzat en aquets projecte és l'algorisme de *Dijkstra*, que permet trobar el camí més curt entre un vèrtex V i tots els altres.

L'objectiu és trobar el camí més curt des d'un vèrtex d'origen a la resta de vèrtexs d'un digraf amb pesos a cada aresta. Per aplicar l'algorisme es necessita un digraf G = (V,E) amb pesos a les arestes i un vèrtex, s, d'origen com a *inputs*.

Per descriure l'algorisme es fixa un ordre dels vèrtexs ($v_0, v_1, v_2 \dots v_n$), el vèrtex s és el primer, $v_0 = s$ i s'introdueix:

- Tres vectors *dist, PreD i S* que tenen tantes components com vèrtexs té el digraf.

- La matriu de pesos que conté la informació de les distàncies.

Per començar s'inicialitzen tots els vectors:

Vector S: El vector S informa dels vèrtexs visitats, cada component és un 0 o un 1. Si la component i-éssima és 1 vol dir que el vèrtex v_i ha estat visitat i si és 0 no ha estat visitat. Per inicialitzar-lo, la primera component és 1 i la resta 0.

Vector dist: La component i-èssima mesura la distància de s a v_i a través dels vèrtexs de S. La component associada a s és 0, les components corresponents a vèrtexs v_i tals que (s, v_i) és un arc, posem el pes de l'arc i la resta de components infinit.

Vector PreD: El vector PreD indica quin és el camí més curt per anar de s a qualsevol altre vèrtex passant pels vèrtexs visitats quan aquest camí existeix. La component i-èssima fa referència al vèrtex anterior en el camí més curt per anar de s a v_i, quan aquest camí existeix. Inicialment és el vector 0.

S'ha de recórrer el conjunt S fins que s'hagin visitat tots els vèrtexs, començant pel que tingui la distància mínima al vèrtex s. Cada vegada que es selecciona un vèrtex, aquest s'afegeix a S ja que està sent visitat.

Una vegada s'ha trobat el vèrtex v_j amb distància mínima, es mira si aquesta distància mínima afegida a cada un dels pesos dels arcs que surten del vèrtex v_j són més petits que els valors que es tenen emmagatzemats en dist. Si resulta que algun valor és més petit, aquest valor s'actualitza a la component corresponent del vector dist i s'afegeix el node des d'on es ve al vector PreD.

S'ha de repetir el procés fins que el vector S sigui tot 1, és a dir fins que s'hagin visitat tots els vèrtexs del graf.

Al final del procés iteratiu s'obté un vector amb les distàncies des del vèrtex s fins a tots els altres i un altre vector PreD amb el vèrtex anterior a cada un en un camí de longitud mínima. Per tant es poden reconstruir tots els camins i saber quant val el seu pes total. D'aquesta manera es té tota la informació emmagatzemada en dos vectors. Un exemple de com es poden reconstruir els camins seria el següent:

Partint de l'exemple exposat en l'apartat anterior *referencia*

	BCN	LON	PAR	OSL
BCN	-	25	-	30
LON	35	-	25	-
PAR	-	40	-	35
OSL	-	35	40	-

Taula 3: Valor dels preus per anar de la ciutat X a Y.

Amb la matriu de pesos corresponents:

$$A = \begin{pmatrix} \infty & 25 & \infty & 30 \\ 35 & \infty & 25 & \infty \\ \infty & 40 & \infty & 35 \\ \infty & 35 & 40 & \infty \end{pmatrix}$$

S'inicialitzen els vectors considerant $v_0 = BCN$; $v_1 = LON$; $v_2 = PAR$; $v_3 = OSL$

$$S = [\,1 \quad 0 \quad 0 \quad 0\,],\ PreD = [\,0 \quad 0 \quad 0 \quad 0\,],\ dist = [\,0 \quad 25 \quad \infty \quad 30\,]$$

Una vegada obtinguts tots els vectors inicialitzats es poden començar a fer les iteracions:

1r iteració:

El node amb pes mínim que encara no ha estat visitat, és a dir S(v) = 0 i dist(v) mínima és el vèrtex 1. Per tant s'agafa com a referència:

S'ha de mirar si des del vèrtex 2 sumant les demès distàncies s'obté algun altre mínim i els vectors quedarien de la següent manera:

$$S = [\,1 \quad 1 \quad 0 \quad 0\,],\ PreD = [\,0 \quad 0 \quad 1 \quad 0\,],\ dist = [\,0 \quad 25 \quad 50 \quad 30\,]$$

2n iteració:

A continuació es troba el vèrtex 3 com a mínim sense haver estat visitat, però no es pot substituir cap mínim ja que les dues arestes sumades, són més grans que el que ja es té.

$$S = [\,1 \quad 1 \quad 0 \quad 1\,],\ PreD = [\,0 \quad 0 \quad 1 \quad 0\,],\ dist = [\,0 \quad 25 \quad 50 \quad 30\,]$$

3r Iteració:

Per últim, s'ha de fer el mateix procés amb el vèrtexs que queden sense visitar, però com en el cas anterior ja es tenen els mínims i no es substitueix res.

$$S = [\,1 \quad 1 \quad 1 \quad 1\,],\ PreD = [\,0 \quad 0 \quad 1 \quad 0\,],\ dist = [\,0 \quad 25 \quad 50 \quad 30\,]$$

En arribar al final de les iteracions, quan el vector S és tot de 1, s'obté el resultat dels dos vectors PreD i dist. Es pot observar que el vector dist dona les distàncies mínimes des de BCN fins a les altres 3 ciutats. El vector PreD diu quina és la ciutat per la qual ha de passar per aconseguir aquesta distància mínima. En tres dels casos tenim 0 al vector PreD, la qual cosa indica que es pot anar directament des de la ciutat d'origen (BCN). Per trobar el camí fins a v_2 s'ha de desfer el camí endarrere. A la ciutat v_2 s'hi ha anat des de la v_1 i anant a la ciutat número v_1 es veu que s'hi ha arribat des de l'origen. Per tant, el camí per arribar a la ciutat v_2 des de la ciutat d'origen és v_0 - v_1 - v_2 (és a dir BCN – LON – PAR) amb una distància, en aquest cas preu, de 50 euros. (Yan)

3.2.2 Altres mètodes

L'algorisme de *Dijkstra* no és l'únic que dóna el camí més curt en una xarxa de vèrtexs. Es poden trobar altres algorismes com ara el de Floyd-Warshall que permet trobar el camí mínim entre dos vèrtexs qualsevols d'una xarxa. És un mètode més general que l'algorisme de *Dijkstra* fet servir, i en el cas d'aquest projecte no interessa saber tots els camins entre tots els nodes ja que es surt un dia en concret i des d'una ciutat en concret i per tant el mètode *Dijkstra* s'adequa més al nostre problema.

Un altre exemple d'algorisme amb el qual es podrien trobar distàncies mínimes en el camp dels grafs és l'algorisme A* que busca la distància mínima entre un vèrtex d'origen i un vèrtex de destí. Cal dir que aquest algorisme es fa servir molt

per calcular distàncies mínimes en xarxes molt grans, que tenen molts nodes i on el recorregut final té molts canvis de nodes. Es fa servir per calcular rutes mínimes com ara en un GPS o en la planificació d'un itinerari per carretera. Es pot trobar una explicació més detallada a *Engineering Route Planning Algorithm* [10].

CAPÍTOL 4. El model

L'objectiu principal del projecte és dissenyar un cercador de vols que permeti oferir paquets de vols que incloguin la visita a 2 o més ciutats durant un període de temps, més o menys definit (dependrà de l'usuari),de tal manera que es pugui obtenir el recorregut més econòmic entre diferents ciutats europees per un determinat nombre de dies. Ara que s'han exposat les bases matemàtiques es pot parlar i adaptar el problema directament amb digrafs. No es parteix d'una idea en la qual l'usuari ha de decidir tots els paràmetres del viatge, ja que aquest és el sistema dels cercadors convencionals: es busquen unes dates predeterminades i un trajecte establert (origen i destí) amb el millor preu i la millor opció de viatge. L'avantatge que proporciona aquest cercador és un augment de la llibertat per a poder a trobar realment el recorregut més econòmic, sense que la destinació sigui una variable definida i sense haver de donar uns dies concrets. Aquesta és una idea innovadora pels joves que habitualment tenen molt de temps lliure a l'estiu i que volen viatjar.

La base dades que s'ha fet servir en aquest projecte conté informació de vols europeus. Com es una base de dades basada en recerques d'altres persones, els preus poden variar en un mateix dia. A més, tots els vols tenen hora de sortida a les 00:00, ja que no disposem de l'hora real de sortida.

De la base de dades s'extrauen tots els preus dels vols entre les ciutats europees. Es busca el preu més baix per cada trajecte (One Way) per cada dia disponible. Per exemple, hi ha un preu per la ruta BCN MAD del dia 02/02/2015, un preu per la ruta MAD BCN del dia 02/02/2015 i així per tots els dies. A mesura que es troba aquesta informació, aquesta s'emmagatzema en una matriu. Es disposa d'un calendari anual, una matriu per blocs on cada bloc correspon a un dia de l'any. El bloc corresponent a cada dia de l'any conté el preu mínim de tots els trams possibles, és a dir, serà una matriu [n x n] on n és el número total d'aeroports dels quals es disposa d'informació.

A partir d'aquí i amb tota la informació ordenada, es pot començar a treballar i aplicar l'algorisme d'optimització.

	BCN	LHR	ORY	FCO	BRU	PRG	VIE
BCN	0	34	34	54	67	76	25
LHR	0	0	45	46	0	45	87
ORY	19	56	0	75	54	76	46
FCO	24	0	65	0	37	0	83
BRU	74	35	0	24	0	56	67
PRG	23	64	34	56	35	0	73
VIE	25	75	0	0	48	27	0

01/01	02/01	03/01	04/01	05/01	06/01	07/01	08/01	09/01	10/01	11/01	12/01	13/01	14/01	15/01	16/01	17/01	18/01	19/01	20/01	21/01	22/01	23/01	24/01	25/01	26/01	27/01	28/01	29/01	30/01	31/01
01/02	02/02	03/02	04/02	05/02	06/02	07/02	08/02	09/02	10/02	11/02	12/02	13/02	14/02	15/02	16/02	17/02	18/02	19/02	20/02	21/02	22/02	23/02	24/02	25/02	26/02	27/02	28/02	0	0	0
01/03	02/03	03/03	04/03	05/03	06/03	07/03	08/03	09/03	10/03	11/03	12/03	13/03	14/03	15/03	16/03	17/03	18/03	19/03	20/03	21/03	22/03	23/03	24/03	25/03	26/03	27/03	28/03	29/03	30/03	31/03
01/04	02/04	03/04	04/04	05/04	06/04	07/04	08/04	09/04	10/04	11/04	12/04	13/04	14/04	15/04	16/04	17/04	18/04	19/04	20/04	21/04	22/04	23/04	24/04	25/04	26/04	27/04	28/04	29/04	30/04	0
01/05	02/05	03/05	04/05	05/05	06/05	07/05	08/05	09/05	10/05	11/05	12/05	13/05	14/05	15/05	16/05	17/05	18/05	19/05	20/05	21/05	22/05	23/05	24/05	25/05	26/05	27/05	28/05	29/05	30/05	31/05
01/06	02/06	03/06	04/06	05/06	06/06	07/06	08/06	09/06	10/06	11/06	12/06	13/06	14/06	15/06	16/06	17/06	18/06	19/06	20/06	21/06	22/06	23/06	24/06	25/06	26/06	27/06	28/06	29/06	30/06	0
01/07	02/07	03/07	04/07	05/07	06/07	07/07	08/07	09/07	10/07	11/07	12/07	13/07	14/07	15/07	16/07	17/07	18/07	19/07	20/07	21/07	22/07	23/07	24/07	25/07	26/07	27/07	28/07	29/07	30/07	31/07
01/08	02/08	03/08	04/08	05/08	06/08	07/08	08/08	09/08	10/08	11/08	12/08	13/08	14/08	15/08	16/08	17/08	18/08	19/08	20/08	21/08	22/08	23/08	24/08	25/08	26/08	27/08	28/08	29/08	30/08	31/08
01/09	02/09	03/09	04/09	05/09	06/09	07/09	08/09	09/09	10/09	11/09	12/09	13/09	14/09	15/09	16/09	17/09	18/09	19/09	20/09	21/09	22/09	23/09	24/09	25/09	26/09	27/09	28/09	29/09	30/09	0
01/10	02/10	03/10	04/10	05/10	06/10	07/10	08/10	09/10	10/10	11/10	12/10	13/10	14/10	15/10	16/10	17/10	18/10	19/10	20/10	21/10	22/10	23/10	24/10	25/10	26/10	27/10	28/10	29/10	30/10	31/10
01/11	02/11	03/11	04/11	05/11	06/11	07/11	08/11	09/11	10/11	11/11	12/11	13/11	14/11	15/11	16/11	17/11	18/11	19/11	20/11	21/11	22/11	23/11	24/11	25/11	26/11	27/11	28/11	29/11	30/11	0
01/12	02/12	03/12	04/12	05/12	06/12	07/12	08/12	09/12	10/12	11/12	12/12	13/12	14/12	15/12	16/12	17/12	18/12	19/12	20/12	21/12	22/12	23/12	24/12	25/12	26/12	27/12	28/12	29/12	30/12	31/12

Figura 12: Representació de la matriu per blocs.

4.1 Dades d'entrada

INPUT: Dades d'entrada: Per començar cal saber quina és la informació que s'obtindrà per part de l'usuari que estigui fent la recerca:

- Ciutat de sortida: Ciutat des de la qual l'usuari vol viatjar, és la ciutat d'on l'usuari surt i torna després del viatge.

- Ciutats a descartar: Si l'usuari ja ha visitat algunes de les ciutats de la llista o simplement no vol anar-hi, té *l'opció* de descartar-les.

- Dies disponibles per viatjar: Es un interval de dies en el qual l'usuari estaria disposat a viatjar.

- Dies de viatge: Dins l'interval de dies disponibles per viatjar que pot ser més gran que els dies de viatge, l'usuari ha de triar un interval de dies que vol estar viatjant. Aquest interval no es correspon a dates absolutes sinó al nombre de dies. Per exemple 6-8, 8-10...

4.2 Construcció del digraf

Una vegada s'han completat tots els paràmetres necessaris, cal organitzar tota la informació per definir la matriu d'adjacència que modela els digrafs sobre la que s'aplicarà l'algorisme d'optimització.

S'ha d'establir un ordre cronològic en els dies per tal de poder organitzar-los bé. S'estableix com a dia 0, el dia en que l'usuari encara es troba a la ciutat d'origen i no ha agafat l'avió. Aquest serà el primer dia de l'algorisme. En el dia 0 només es pot anar des de la ciutat d'origen a les altres ciutats. A partir d'aquí es troben els dies 1, 2, 3, 4... que seran els dies següents. Quan l'algorisme determini en quin dia comença el viatge, es podrà saber el dia en que l'usuari sortirà de la ciutat d'origen. Per exemple, si la ciutat d'origen és Barcelona i el primer destí és Paris, i a més l'usuari viatja el dia 1 voldrà dir que l'algorisme diu que viatjarà des

de Barcelona al dia 0 (BCN0) fins a Paris al dia 1 (PAR1). És a dir, cal fixar-se sempre en el dia de la segona ciutat, ja que aquest serà el dia en que l'usuari viatjarà. Es pot veure un exemple de com estan distribuïdes les ciutats i els dies en la matriu:

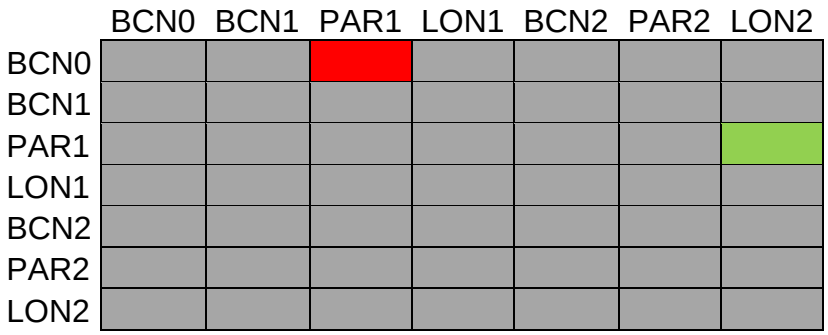


Figura 13: Exemple de la relació entre dies i ciutats en la matriu d'adjacència.

En la casella vermella s'hi troba el preu corresponent a viatjar des de Barcelona fins a Paris el dia 1 del viatge i a la casella verda s'hi troba el preu corresponent a viatjar des de Paris fins a Londres el dia 2 del viatge.

Una vegada establerta la classificació dels dies, es pot passar a explicar l'estructura que tindrà el període de temps en el que l'usuari vol viatjar (Dies de viatge). Per fer-ho es fragmenta la matriu d'adjacència en 3 fases o trossos. El primer fragment correspondrà als dies de sortida, en els quals l'usuari agafarà el seu primer vol sent l'origen la seva ciutat de sortida. Aquesta matriu de la primera fase només tindrà *l'opció* de volar des de la ciutat de sortida fins a totes les altres ciutats, per tant s'obtindrà un vector fila. Aquest vector està compost pels diferents preus des de la ciutat de sortida fins a totes les altres ciutats, organitzats d'acord amb els possibles dies de sortida. Per exemple, assumint que l'usuari té dos possibles dies de sortida, aquest vector es forma a partir de l'encadenament dels dos vectors que donen el preu de volar des de la ciutat de sortida a la resta, en els dos dies triats.

Figura 14: Exemple de la matriu corresponent a la fase 1 del model.

El color verd indica els preus de volar de la ciutat de sortida a la resta el dia 1, i el vector blau ens dóna la mateixa informació, però referida al dia 2.

El segon fragment és el cos de la matriu, representa tots els dies que l'usuari està viatjant. Per formar aquesta matriu s'han d'agafar totes les matrius dels dies en que l'usuari viatja i ajuntar-les. Existeixen diferents maneres d'ajuntar les matrius per tal d'aconseguir aquest segon fragment. Es disposa d'un nombre determinat de dies a la segona fase, com ara el número 6. En aquest cas, el

primer bloc que es realitza és de 6 (d'aquesta manera, s'assumeix que cada dia es pot canviar de ciutat):

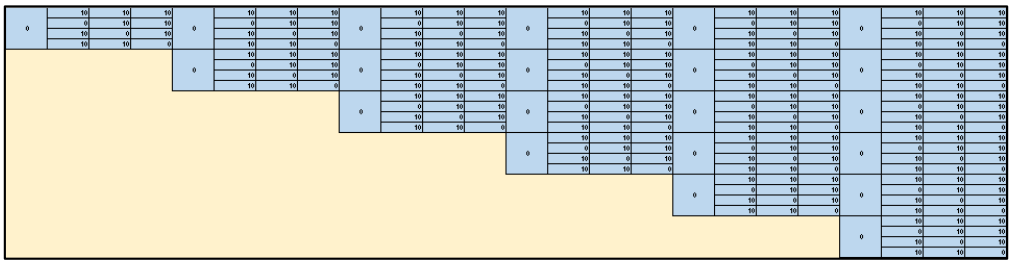


Figura 15: Exemple de la matriu corresponent a la fase 2 del model amb 1 bloc.

En aquest segon fragment cal destacar que la primera columna i la diagonal de cada matriu sempre seran 0, ja que la primera columna correspon a la ciutat de sortida (i no s'hi ha de tornar abans d'hora) i la diagonal fa referència a anar d'una ciutat a ella mateixa. Les matrius d'un mateix bloc columna són la mateixa ja que fan referència a viatjar un dia en concret a una ciutat i des d'una ciutat en concret.

Aquesta matriu, que és un bloc sencer, permet viatjar a 2 ciutats diferents com a mínim, ja que en la primera fase l'usuari va a una ciutat, en la segona fase es desplaça a una ciutat diferent i, com s'explicarà més endavant, en una tercera fase l'usuari torna a l'origen. Ara bé, podria passar que segons aquest model, es destini només un dia a la primera ciutat i 5 dies a la segona. Com que sembla raonable destinar un mínim de dos dies a cada ciutat i les ciutats no es coneixen a priori, es proposen altres models de matriu per aquesta segona fase. Això s'aconsegueix dividint els dies n que l'usuari disposa per viatjar, en aquest cas essent de 6 (dies de viatge) entre diferents nombres r, $r_i = 1,2,3 \ldots$ tals que $\frac{n}{r} \geq 2$ .D'aquesta manera es defineix un vector que té r components.

Si resulta que la suma de totes les components del vector no sumen els n dies de viatge, es va afegint 1 des de la posició del mig i cap a les bandes (Si el vector és parell es comença amb la posició del mig més propera a l'inici).

Per exemple, en el cas de 6 dies de viatge, s'obtenen els vectors següents:

[6] [3 3] [2 2 2]

Ja s'ha vist anteriorment la matriu corresponent al primer, i ara es poden veure les matrius corresponents al segon i al tercer vector:

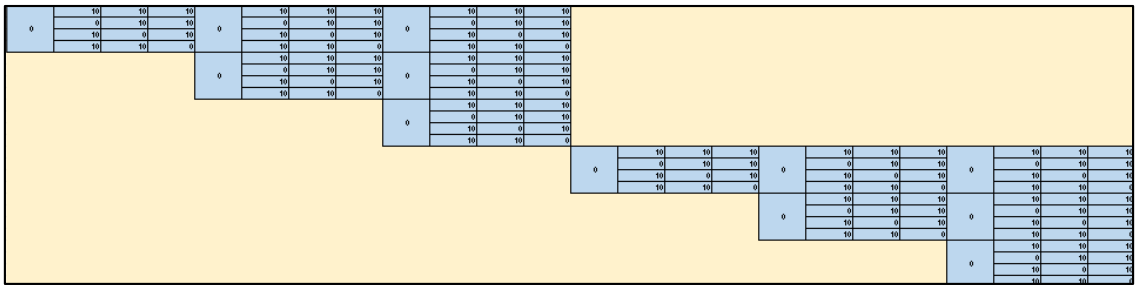


Figura 16: Exemple de la matriu corresponent a la fase 2 del model amb 2 blocs.

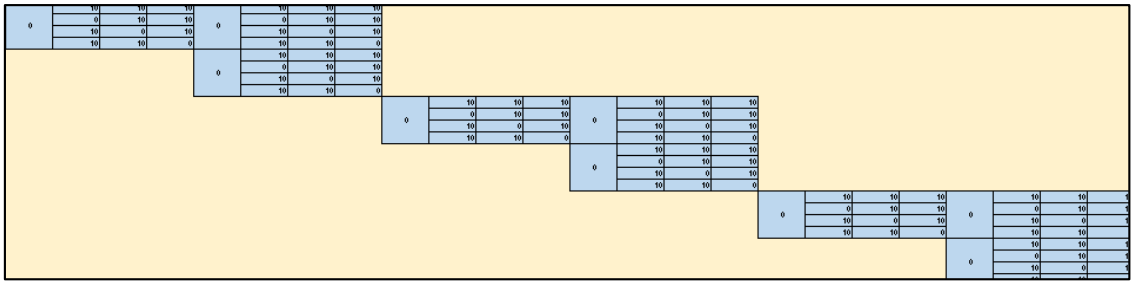


Figura 17: Exemple de la matriu corresponent a la fase 2 del model amb 3 blocs.

En el cas del vector amb dues components es visitarien 3 ciutats i en el cas del vector amb tres components es visitarien 4 ciutats.

La segona fase pot tenir qualsevol d'aquests formats, de fet més endavant s'exposa un procés iteratiu en el qual es busca la millor opció en relació al número de ciutats visitades i al preu. Així es pot oferir un ampli ventall d'oportunitats a l'usuari per a que pugui triar.

Finalment en la tercera fase les matrius només tindran preus a la columna de la ciutat d'arribada. Així s'obliga l'usuari a tornar a la ciutat d'on ha sortit. Per exemple, en el cas plantejat, assumint que l'usuari té dos possibles dies d'arribada, la matriu de la fase 3 serà així:

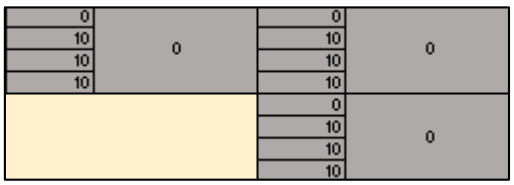


Figura 18: Exemple de la matriu corresponent a la fase 3.

4.3 Execució del Procés

Si s'ajunten les matrius de totes les fases s'obté la matriu amb la qual es pot treballar i que actuarà de matriu de pesos del digraf que modela el problema i sobre el que s'aplicarà l'algorisme de *Dijkstra*.

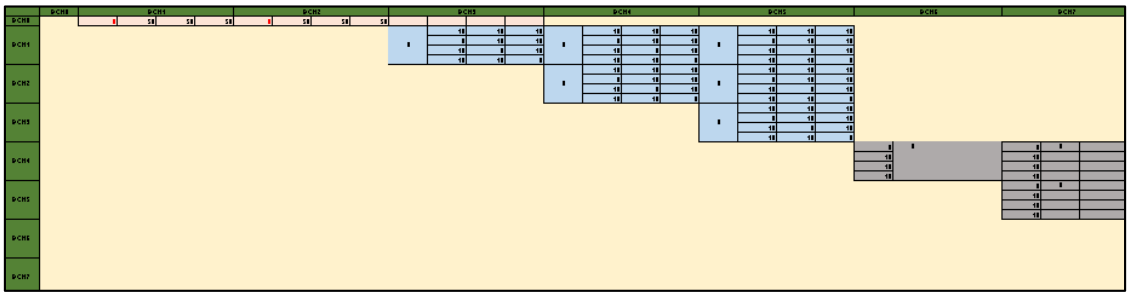


Figura 19: Exemple de la matriu corresponent a les 3 fases juntes, matriu d'adjacència on s'aplica *Dijkstra*.

Suposem que n és el número de dies que pot durar el viatge. Cada vegada que es fa el procés de mirar un interval de dies, per exemple sortir el dia 1 i tornar el dia 8 (amb un *input* de 8 dies disposats a viatjar) s'ha de construir la matriu de pesos i aplicar el *Dijkstra* n/2 vegades, arrodonint a la baixa, ja que es proposa fer el procés amb tots els casos de fase 2. Per exemple en el cas de 8 dies viatjant, es farien 4 alteracions de la fase 2 que serien les següents:

Iteració	Format de la Fase 2
1	[8]
2	[4 4]
3	[2 3 3]
4	[2 2 2 2]

Taula 4: Exemple de la dimensió dels blocs de la fase 2 per a cada iteració.

A més de fer aquest primer tipus d'iteracions, cal saber quants dies disponibles té l'usuari per viatjar, ja que es proposa iterar el procés sencer més vegades. Per exemple, si l'usuari està disposat a viatjar 8 dies però té disponibilitat per viatjar 15 dies s'haurà de iterar el procés fins a cobrir aquest marge de 15 dies.

Iteració	Dia Inicial	Dia Final
1	1	8
2	2	9
3	3	10
4	4	11
5	5	12
6	6	13
7	7	14
8	8	15

Taula 5: Exemple d'interval de dies de viatge per a cada iteració.

Amb el resultat de *Dijkstra* s'obtenen les diferents ciutats per les quals passarà l'usuari així com els dies en que ho farà. Per altra banda també s'obtindrà el preu total del trajecte.

4.4 Output: El resultat del procés

Després de repetir totes les iteracions es proposen moltes opcions de viatge. Cada opció conté un itinerari per diferents ciutats amb els dies de vol i el preu total. Com més disponibilitat tingui l'usuari per a viatjar, més opcions s'obtindran. Aquestes opcions es presenten emmagatzemades en 3 matrius, totes amb el mateix nombre de columnes. Cada columna correspon a una opció de viatge:

-Matriu Preu: És un vector fila que conté el preu total de cada opció de viatge.

-Matriu Ciutats: Cada columna conté les ciutats que es visiten, en ordre cronològic.

-Matriu Dies: Cada columna conté els dies que l'usuari fa cada salt. Per exemple si el primer salt és BCN MAD el dia que hi hagi a la matriu correspondrà al dia que anirem des de BCN a MAD.

78	76	161	139	126	131	80
BCN	BCN	BCN	BCN	BCN	BCN	BCN
MIL	DUB	HAM	ROM	MIL	CLJ	MAN
BRU	BRU	GVA	STR	RIX	GVA	DUB
BCN	BCN	BRU	BCN	BCN	BCN	BCN
		BCN				
06/02/2015	07/02/2015	09/02/2015	10/02/2015	11/02/2015	12/02/2015	14/02/2015
10/02/2015	11/02/2015	11/02/2015	14/02/2015	15/02/2015	16/02/2015	18/02/2015
11/02/2015	12/02/2015	13/02/2015	15/02/2015	16/02/2015	17/02/2015	19/02/2015
		14/02/2015				

Taula 6: Exemple de les 3 matrius que s'obtenen com a resultat.

Per exemple, la columna 2 de les matrius anteriors correspon a l'opció més econòmica, costa 76 euros, ens permet viatjar de BCN a DUB el dia 07/02/2015, de DUB a BRU el dia 11/02/2015 i tornar a BCN des de BRU el dia 12/02/2015.

Els resultats s'exporten a un arxiu *Excel* per ordenar-los i perquè l'usuari pugui triar el que més li convingui. L'exportació a *Excel* és un mètode alternatiu a la pàgina web. Una vegada exportats es poden filtrar segons el preu del viatge, per alguna ciutat en concret o segons els dies de sortida. D'aquesta manera, l'usuari pot visualitzar els resultats de la manera que més li convingui.

4.5 Rectificacions del model

4.5.1 Repetició de ciutat

S'ha afinat al màxim el model per tal de poder arribar a una solució que compleixi tots els objectius que s'han plantejat. Amb el model proposat hi ha casos en que l'usuari podria no acceptar el viatge, ja que es pot repetir més d'una ciutat. Aquesta repetició de ciutats s'arregla eliminant les solucions en les que es va a una ciutat i al llarg del viatge s'hi torna per segona vegada. Aquest problema només sorgeix en viatges llargs i amb molts salts i per tant són els més cars i els que menys interessen a l'usuari. A més a més tenint en compte que el número de solucions que s'obté és alt i que aquest problema només passa en la minoria, s'opta per eliminar aquestes solucions.

4.5.2 Exclusió de ciutats

S'ha afegir una opció addicional que millora el programa: l'usuari podrà eliminar al principi de tot el procés les ciutats que ja hagi visitat, o bé on no vulgui anar. L'objectiu d'aquesta opció no és eliminar un gran volum de ciutats, sinó eliminar aquelles poques ciutats que ja s'han visitat o que realment no es volen veure. Una vegada l'usuari hagi triat aquestes ciutats, s'eliminaran directament de la matriu de pesos de tal manera que serà impossible arribar-hi. Aquesta opció també es pot fer servir per fer una segona recerca, si els primers resultats no convencen l'usuari. En aquest cas el preu pujarà ja que no serà el mínim però les ciutats poden resultar més atractives per l'usuari. Aquest últim haurà de valorar, un cop fetes les dues recerques, si val la pena pagar més per realitzar alguns dels nous itineraris.

CAPÍTOL 5. Descripció de Programació

5.1 Base de Dades

La base de dades amb la que hem treballat prové de resultats de recerques fetes amb SkyScanner, un dels cercadors de vols que s'ha explicat al primer capítol. Per tant les dades amb les quals s'ha treballat són resultats de diferents recerques en aquesta pàgina web. És un dels cercadors més destacats actualment i on el nombre de recerques realitzades és molt gran, fet que permet tenir molta informació per a processar.

Per tal de poder treballar i desenvolupar el programa, s'ha partit des d'una base de dades en format *.sql* que conté la informació d'uns 450.000 vols europeus del mes de febrer de 2015. Aquesta base de dades conté el número de vol, origen, destí, data de sortida i preu del bitllet, però fixant tots els vols de sortida a les 00h00.

5.2 Funcions Bàsiques

5.2.1 Lectura de Dades

S'ha creat un funció que llegeix totes les dades necessàries de la base de dades. Primer de tot cal obrir la base de dades, importar-la a *Matlab* i definir cada columna, tenint en compte que les dades per files estan delimitades amb comes. Cada dada que es llegeix cal transformar-la segons sigui *string* o sigui un número. Una vegada importades totes les dades al *matlab,* amb el format corresponent, s'obtindrà una llista *PriceList*, amb tota la informació necessària.

Aquesta *PriceList* tindrà diferent informació: nº de vol, Origen, Destí, Preu, Data Sortida, Data Arribada. Una vegada obtinguda la *PriceList* es pot començar a treballar amb les dades.

5.2.2 AirportList

Una vegada s'ha llegit i carregat la llista de preus al *Matlab,* s'extreu una llista de tots els aeroports amb els que es treballarà. La funció *OptimizeAirportListV2* recorre tota la *PriceList* i es queda solament amb els aeroports dels quals es tenen dades. D'aquesta manera s'evita que la llista contingui informació d'aeroports dels quals no tenim dades. Per tal de poder dur a terme l'objectiu, l'*input* de la funció serà la *PriceList* i el seu *output* serà l'*AirportList*, un vector amb el llistat de tots els aeroports en codi IATA (3 lletres).

5.2.3 FromTo

Una vegada obtinguts tots els aeroports ordenats en un vector, es pot passar a ordenar totes les dades de preus que es troben a la *PriceList*. A partir d'aquí es

construeix la matriu per blocs amb la qual es treballarà la resta del temps. *FromTo* serà l'*output* de la funció GetPriceMatrixForDayV2 i serà una matriu de cel·les de dimensió 12x31, és a dir tindrà la dimensió d'un any, cada fila correspondrà a un mes i cada columna correspondrà a un dia.

Per tal de poder ordenar bé les dades cada cel·la dia tindrà una matriu al seu interior de dimensions [n x n] on n és la longitud del vector *AirportList*. Els dies que no existeixen en el calendari, com per exemple el 31 de juny tindrà un valor assignat de 0 en la matriu, que correspon a la matriu nul·la.

Una vegada definida l'estructura d'aquesta matriu la funció recorrerà tota la llista de preus. Caldrà fixar-se en el dia, l'origen, el destí i el preu de cada fila. Primer s'ha de buscar a la matriu de cel·les el dia en concret, i una vegada trobada la matriu corresponent a aquest dia s'haurà de mirar si el preu de la casella [origen x destí] és 0.Si és igual a 0 es posarà el preu de la llista, i si en canvi és diferent a 0 només el es substituirà si és menor.

L'*output* final d'aquesta funció representa els preus mínims de cada dia per a cada ruta de la base de dades utilitzada, de tal manera que es poden trobar les dades necessàries de forma molt ràpida en una matriu.

5.2.4 DepartureCityToFirst

Les matrius guardades en els diferents dies de l'any tenen totes el mateix ordre d'aeroports, que és el de l'*AirportList*. Per a poder treballar millor, quan es munten les matrius de pesos es fa servir la funció *DepartureCityToFirst* per posar sempre la ciutat de sortida el número 1 del vector *AirportList*, i d'aquesta manera a partir de llavors no cal buscar-la cada vegada, ja que sempre serà la que estigui en la posició 1. Els *inputs* d'aquesta funció són una matriu de preus diària i l'aeroport d'origen, per poder així retornar aquesta matriu de preus diària amb l'aeroport d'origen en primera posició.

5.2.5 Funcions Aeroport

Per tal de poder treballar amb la llista d'aeroports obtinguts de la base de dades cal crear una relació numèrica entre els aeroports. Quan es llegeix la base de dades els aeroports s'ordenen a mida que van apareixent en un vector. El número al qual correspondrà cada aeroport a partir d'aquell moment és la seva posició en el vector. D'aquesta manera es podrà manejar molt més fàcilment. El nom de l'aeroport sempre està en codi IATA, és a dir, un codi de tres lletres que representa cada aeroport.

Per aquesta raó s'han creat dues funcions que transformen el número de l'aeroport en nom i viceversa.

GetAirportName: L'*input* d'aquesta funció és un número del 1 fins a la llargada del vector *Airport* i el seu *output* és el nom de l'aeroport. Si el número no es troba

en el rang que requereix la funció, un missatge d'error avisa que aquest número no és correcte.

GetAirportNum: L'*input* d'aquesta funció és el nom de l'aeroport en codi IATA, que, com s'ha dit abans, està composat de tres lletres. La funció retornarà la posició en el vector d'aeroports d'aquest aeroport, per tal de poder treballar còmodament. Si el número de caràcters és diferent de tres o si no hi ha cap aeroport corresponent al codi entrat, la funció retornarà el corresponent missatge d'error.

5.2.6 Funcions Dates

Com en el cas dels aeroports, treballar amb les dates senceres, dia, mes i any és molt incòmode. Cal crear una funció bijectiva que associï un número a cada dia de l'any. Es pren com a dia 1 el dia 1/1/2015, a partir d'aquí cada dia que passi se li sumarà 1. Independentment del dia, mes o any que sigui, tots estaran referenciats en el mateix punt de partida.

Es creen doncs dues funcions per tal de poder dur a terme aquesta transformació. La primera serà per passar de data a número i la segona per poder fer el camí invers.

GetDayNumber: Els *inputs* d'aquesta funció seran el dia, el mes i l'any que l'usuari desitgi i donaran el número amb el qual es podrà treballar. En el cas que el format dels números no sigui el correcte, la funció mostrarà un missatge d'error indicant que hi ha un error.

GetDayDate: L'*input* d'aquesta funció serà el número amb el qual s'estigui treballant per tal de poder donar un resultat visual a l'usuari amb el detall del dia, mes i any prèviament assenyalats. Per tant l'*output* serà la data sencera. Si el número no és positiu o no té un format adequat es rebrà el missatge d'error corresponent.

5.3 Funcions Complexes

5.3.1 MultiDestinationMatrixFlexV6

L'objectiu de la funció *MultiDestinationMatrixFlexV6* és muntar la matriu de pesos sobre la que s'aplica l'algorisme de *Dijkstra*. És una de les funcions més complexes del programa ja que depèn de moltes variables. Els seus *inputs* són:

-Phase1Days: Número de dies que es vol que duri la fase 1.

-Phase2Days: Número de dies que es vol que duri la fase 2. Pot ser també en forma de vector, ja que la fase 2 pot estar separada de diferents maneres.

-Phase3Days: Número de dies que es vol que duri la fase 3.

-DepartureCity: Ciutat des de la qual l'usuari sortirà.

-DepartureDay: Dia de sortida de l'usuari o dia d'inici del viatge.

-DepartureMónth: Mes de sortida de l'usuari o mes d'inici del viatge.

-DepartureYear: Any de sortida de l'usuari o any d'inici del viatge.

-FromTo: La matriu per blocs que conté tota la informació necessària relacionada amb els preus.

-NoVisit: Un *input* extra que dona l'opció de descartar ciutats ja visitades o on no s'hi vol anar.

Una vegada es tenen tots els paràmetres d'entrada, es pot veure què fa la funció per tal de treure la matriu de pesos.

Primer de tot, la funció calcula la suma dels dies que l'usuari està de viatge per extreure la informació de la matriu *FromTo* i posar-la de banda. D'aquesta manera es redueix la mida de la matriu amb la que treballem i només queden els dies que interessen.

Si *NoVisit* és diferent de 0, vol dir que s'ha de descartar alguna ciutat i per tant, de les matrius que farem servir, posarem 0 a totes les columnes de les ciutats on l'usuari hagi decidit que no vol anar. D'aquesta manera es bloqueja l'accés a aquests aeroports.

Ara ja només queden les dades que realment es volen fer servir. S'han de muntar les tres fases com s'ha explicat anteriorment. Per muntar la primera fase només es fa servir la primera fila de les matrius de la fase 1 ja que és la ciutat de sortida i no calen més dades. Per a fer la fase dos es col·loquen les matrius en blocs i depenent de la mida del vector, hi hauran tants blocs de la fase 2 com números componguin el vector. Per acabar, per muntar la fase 3 només cal la columna de la ciutat d'origen, que serà la mateixa que la de destí, i a tota la resta s'hi introduirà 0 perquè el viatge no pugui acabar en cap altre ciutat.

Una vegada muntades les tres fases per separat, es poden ajuntar i posar 0 als espais buits que quedin.

5.3.2 *Dijkstra* Algorithm

La funció *Dijkstra*Algorithm segueix el procés de l'algorisme de *Dijkstra* que s'ha explicat en el capítol 3. La dada d'entrada que cal aportar és la matriu de pesos construïda amb la funció *MultiDestinationMatrixFlexV6,* que hem introduït en la secció anterior. Les dades de sortida que proporcionarà la matriu, són un vector amb els diferents preus per a cada camí i un altre vector amb l'itinerari corresponent a cada preu.

5.3.3 ResultatsVisualV2

Una vegada obtinguts els dos vectors amb la funció de l'algorisme de *Dijkstra* s'han d'interpretar aquests resultats per poder obtenir els diferents dies del viatge així com les diferents ciutats per les que l'usuari passarà. Aquesta funció necessita els següents paràmetres d'entrada:

-DepartureDate: La data de sortida, en format numèric partint del 1/1/2015.

-AirportList: La llista d'aeroports, per poder relacionar els números dels aeroports amb els noms.

-DepartureCity: La ciutat de sortida.

-PreD: El vector resultat de *Dijkstra* que indica el camí pel qual l'usuari ha passat.

-Distance: El vector resultat de *Dijkstra* que indica les distàncies als diferents punts.

-Phase1Days: Número de dies de la fase 1.

-Phase3Days: Número de dies de la fase 3.

L'objectiu de la funció és retornar els dos vectors resultats de *Dijkstra* de manera que es pugui visualitzar i veure bé quin ha estat el recorregut que s'ha seguit i per quines ciutats s'ha passat.
Se sap que el punt final del viatge sempre succeeix en els dies de la fase 3, per tant s'ha de començar des del final per veure quin camí s'ha seguit. Per saber quina posició del vector correspon a cada dia i a cada aeroport, s'ha de dividir el vector entre el número de posicions de la *AirportList.* D'aquesta manera queden definits diferents blocs. Cada bloc correspon a un dia i a cada dia hi haurà tots els aeroports.

0	0	0	0	0	0	0	2	0	7	0	0	0

Figura 20: Exemple de vector PreD

BCN	BCN	LON	PAR	OSL	BCN	LON	PAR	OSL	BCN	LON	PAR	OSL
0	0	0	0	0	0	0	2	0	7	0	0	0
Dia 0			Dia 1				Dia 2				Dia 3	

Figura 21: Exemple de la divisió de blocs

Pel que fa al vector dist, en relació als dies corresponents a la fase 3, és a dir els últims, i mirant les posicions del vector corresponents a l'aeroport de origen/destí s'hi pot trobar el preu total del trajecte i la distància total que s'ha recorregut per arribar al destí l'últim dia.

0	9999	10	9999	9999	9999	9999	35	9999	59	9999	9999	9999

Figura 22: Exemple vector dist on 9999 és el valor que substitueix l'∞.

Per saber tant el recorregut com els dies en que l'usuari surt de cada ciutat, cal centrar-se en el vector PreD. Si es comença des del final, des de la ciutat i dia d'arribada que ja són coneguts, es pot trobar la posició del vector de la qual s'ha vingut. A aquesta posició li correspondrà un dia i un aeroport que pot saber-se mitjançant la divisió inicial. Caldrà repetir aquest procés fins que s'arribi a l'aeroport i dia de sortida.

BCN→ Dia 1 → LON → Dia 2 → PAR → Dia 3 → BCN = **59 euros**

La funció repeteix aquest procés i els seus *outputs* són:
-Vector amb les ciutats que l'usuari ha visitat
-Vector amb els dies en que l'usuari ha sortit de cada ciutat

5.3.4 Main Iterative

Aquesta funció engloba tot el procés del programa i seria el que hauria d'anar a una possible pàgina web on es pugués desenvolupar el programa. Primer de tot la funció demana a l'usuari les dades d'*input* que es necessiten per dur a terme tot el procés:

-Dia d'inici de la disponibilitat de l'usuari

-Dia final de la disponibilitat de l'usuari

-Quina serà la ciutat d'origen / destí de l'usuari

-Dies mínims/màxims que es vol estar viatjant. Aquesta opció vindria en un desplegable, per exemple: 6-8, 8-10, 6-10...

Amb aquesta informació el programa en té prou per oferir un ampli ventall d'oportunitats de viatge a l'usuari. Es realitzarà un procés iteratiu durant tots els dies de disponibilitat. En particular dins el marge de disponibilitat això inclou:

-La creació de la matriu de pesos de manera iterativa també, amb totes les seves opcions.

-L'aplicació de l'algorisme de *Dijkstra* amb aquestes matrius de pesos.

-La visualització de resultats de totes les vegades que s'ha executat l'algorisme de *Dijkstra*.

I per últim caldrà ajuntar totes les dades obtingudes en vectors per poder mostrar els resultats obtinguts. Una vegada obtinguts tots els resultats, s'eliminen aquells en que es repeteixi una ciutat més d'una vegada (excepte la ciutat d'origen/destí) ja que en un viatge l'usuari no vol visitar dues vegades la mateixa ciutat. Aquests casos passen en viatges en que es visiten moltes ciutats i per tant els menys econòmics.

Quan tenim les dades finals es poden exportar a un arxiu Excel per tal de processar-les i veure quina opció de viatge pot interessar més l'usuari.

5.3.5 ExportData

La funció *Export Data* permet exportar els resultats finals a un full de *Excel* per tal de veure'ls millor i poder triar la millor opció de viatge. El programa de *Matlab* proporcionarà finalment tres matrius:

-Price: Un vector amb preus segons l'opció de viatge que l'usuari hagi triat

-TravelDays: Una matriu on les columnes són les diferents opcions de viatge i les files són els diferents dies en els quals l'usuari hauria d'agafar un vol.

-TravelCities: Una matriu on les columnes són les diferents opcions de viatge i les files són les diferents ciutats que l'usuari visitarà.

Les tres matrius s'exporten a l'*Excel* de tal manera que a cada columna s'hi troben els preus, els dies que l'usuari estarà de viatge i les ciutats que visitarà. A partir d'aquí cada usuari pot filtrar el que més li interessi per veure quin és el viatge que s'adequa més als seus interessos. Així també es poden veure les diferències de preu entre visitar 3 o 4 ciutats per exemple i valorar si es vol pagar més i visitar alguna ciutat addicional o no. L'usuari és lliure d'escollir així *l'opció* que més li convingui, tal com hem vist a la Taula 6.

A més, en el mateix *Excel* s'ha creat una *macro* que ordena les opcions de viatja en ordre cronològic o en ordre de preus, depenent del que l'usuari vulgui. Aquesta opció d'ordenar les diferents opcions són també les que es posarien en la pàgina web. Així que l'usuari veuria tots les opcions possibles, ordenades de més interesants al seu criteri, a menys interesants.

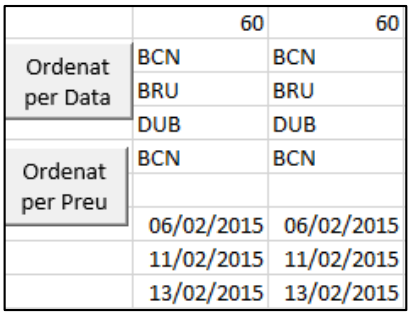


Figura 23: Botons de l'*Excel* per on s'ha d'executar la macro.

CAPÍTOL 6. Resultats

6.1 Resultats Obtinguts amb el programa

Una vegada executat el programa, s'obté el document d'*Excel* on es poden veure les diferents opcions de viatge incloent el preu, l'itinerari i els dies de vol. Es pot veure com a partir de les dades d'entrada s'han obtingut resultats consistents. Els resultats es poden classificar segons la demanda de l'usuari. La primera opció és ordenar-los per preu i també es poden buscar tots els viatges que continguin una ciutat en concret per veure els diferents preus o ordenar-los per dates. A continuació es mostren diferents exemples de recerques i els resultats que s'han pogut obtenir.

6.2 Exemple

En el primer exemple les dades d'entrada que l'usuari donarà són les següents:

- Dia d'inici de disponibilitat: 02/02/2015
- Dia final de disponibilitat per viatjar: 15/02/2015
- Ciutat des de la qual volem viatjar: BCN
- Màxim de dies que dura el viatge: 7
- Mínim de dies que dura el viatge: 9

```
>> MainIterativeTestV2
Set Start of Disponibility
02/02/2015
Set End Day of Disponibility
15/02/2015
From which city do you want to start your travel?
BCN
Minimum Traveling Days
7
Maximum Traveling Days
9
```

Figura 24: Dades d'entrada del programa per a l'exemple 1.

Amb les dades d'entrada que demana el programa es pot obtenir la solució exportada a l'*Excel*. Un fragment de la solució que s'ha obtingut ordenant cronològicament els dies de sortida és la que es pot veure a la Figura 25. Només s'ha posat un fragment de la solució ja que aquesta té 47 possibles opcions.

63	77	86	114	63	86	91	60
BCN	BCN	BCN	BCN	BCN	BCN	BCN	BCN
ROM	ROM	MIL	ROM	ROM	MIL	ROM	BRU
BRU	BRU	CLJ	BRU	BRU	CLJ	CTA	DUB
BCN	BCN	GVA	DUB	BCN	GVA	GVA	BCN
		BCN	BCN		BCN	BCN	
04/02/2015	04/02/2015	04/02/2015	04/02/2015	04/02/2015	04/02/2015	04/02/2015	06/02/2015
08/02/2015	08/02/2015	06/02/2015	06/02/2015	10/02/2015	06/02/2015	08/02/2015	11/02/2015
11/02/2015	10/02/2015	09/02/2015	08/02/2015	12/02/2015	09/02/2015	10/02/2015	13/02/2015
		11/02/2015	10/02/2015		11/02/2015	12/02/2015	

Figura 25: Resultat exportat directament a l'*Excel* de l'exemple 1

Una vegada exportat el resultat a l'*Excel* es poden ordenar les opcions de més econòmica a menys econòmica. També poden visualitzar-se de manera més atractiva, ressaltant el preu:

Opció	1	Opció	2	Opció	3	Opció	4	Opció	5	Opció	6
Preu	60	Preu	60	Preu	63	Preu	63	Preu	63	Preu	68
Ciutat	BCN	Ciutat	BCN	Ciutat	BCN	Ciutat	BCN	Ciutat	BCN	Ciutat	BCN
Dia	06/02/2015	Dia	06/02/2015	Dia	04/02/2015	Dia	04/02/2015	Dia	05/02/2015	Dia	08/02/2015
Ciutat	BRU	Ciutat	BRU	Ciutat	ROM	Ciutat	ROM	Ciutat	ROM	Ciutat	LON
Dia	11/02/2015	Dia	11/02/2015	Dia	08/02/2015	Dia	10/02/2015	Dia	10/02/2015	Dia	13/02/2015
Ciutat	DUB	Ciutat	DUB	Ciutat	BRU	Ciutat	BRU	Ciutat	BRU	Ciutat	DUB
Dia	13/02/2015	Dia	13/02/2015	Dia	11/02/2015	Dia	12/02/2015	Dia	12/02/2015	Dia	15/02/2015
Ciutat	BCN	Ciutat	BCN	Ciutat	BCN	Ciutat	BCN	Ciutat	BCN	Ciutat	BCN

Figura 26: Primera part dels resultats ordenats per preus

En aquest exemple en particular es troben resultats de 60 euros per a viatges de dues ciutats com es pot veure en la figura 26. Algunes de les ciutats que s'ofereixen són Brusel·les, Dublin, Roma o Londres, grans ciutats turístiques europees per a visitar.

Si s'avança una mica més i es miren les altres opcions disponibles el preu augmenta però hi ha més varietat de ciutats i també s'incrementa la possibilitat de viatjar més dies.

Opció	9	Opció	10	Opció	11	Opció	12	Opció	13
Preu	86	Preu	86	Preu	89	Preu	91	Preu	91
Ciutat	BCN	Ciutat	BCN	Ciutat	BCN	Ciutat	BCN	Ciutat	BCN
Dia	04/02/2015	Dia	04/02/2015	Dia	06/02/2015	Dia	04/02/2015	Dia	05/02/2015
Ciutat	MIL	Ciutat	MIL	Ciutat	DUB	Ciutat	ROM	Ciutat	ROM
Dia	06/02/2015	Dia	06/02/2015	Dia	11/02/2015	Dia	08/02/2015	Dia	08/02/2015
Ciutat	CLJ	Ciutat	CLJ	Ciutat	BRU	Ciutat	CTA	Ciutat	CTA
Dia	09/02/2015	Dia	09/02/2015	Dia	14/02/2015	Dia	10/02/2015	Dia	10/02/2015
Ciutat	GVA	Ciutat	GVA	Ciutat	BCN	Ciutat	GVA	Ciutat	GVA
Dia	11/02/2015	Dia	11/02/2015			Dia	12/02/2015	Dia	12/02/2015
Ciutat	BCN	Ciutat	BCN			Ciutat	BCN	Ciutat	BCN

Figura 27: Segona part dels resultats ordenats per preus

Si es comparen les opcions 1 a 6 de la figura 26 amb les opcions 9 a 13 de la figura 27 es pot apreciar un lleu increment del preu i en canvi es pot observar un canvi en les ciutats obtingudes. El viatge en aquest cas s'incrementa uns dies per un augment de només 25 euros respecte els preus més econòmics.

En l'exemple anterior, suposant que s'elimina la ciutat de Dublin, com si l'usuari ja hi hagués anat o no volgués visitar-la, es pot demostrar que *l'opció* que incorpora el buscador d'eliminar ciutats funciona correctament.

Opció	1	Opció	2	Opció	3	Opció	4	Opció	5	Opció	6
Preu	63	Preu	63	Preu	63	Preu	63	Preu	77	Preu	84
Ciutat	BCN	Ciutat	BCN	Ciutat	BCN	Ciutat	BCN	Ciutat	BCN	Ciutat	BCN
Dia	04/02/2015	Dia	04/02/2015	Dia	05/02/2015	Dia	05/02/2015	Dia	04/02/2015	Dia	08/02/2015
Ciutat	ROM	Ciutat	ROM	Ciutat	ROM	Ciutat	ROM	Ciutat	ROM	Ciutat	LON
Dia	08/02/2015	Dia	10/02/2015	Dia	10/02/2015	Dia	10/02/2015	Dia	08/02/2015	Dia	13/02/2015
Ciutat	BRU	Ciutat	BRU	Ciutat	BRU	Ciutat	BRU	Ciutat	BRU	Ciutat	BFS
Dia	11/02/2015	Dia	12/02/2015	Dia	13/02/2015	Dia	12/02/2015	Dia	10/02/2015	Dia	15/02/2015
Ciutat	BCN	Ciutat	BCN	Ciutat	BCN	Ciutat	BCN	Ciutat	BCN	Ciutat	BCN

Figura 28: Resultat eliminant DUB

Els resultats obtinguts després d'eliminar Dublin suposen un increment del preu, ja que els preus mínims suposaven passar per Dublin. Es pot observar que aquestes ofertes han estat substituïdes per altres que en la Figura 26 no apareixien.

CAPÍTOL 7. Conclusions

El programa que s'ha dissenyat compleix amb les expectatives inicials que s'havien marcat. El cercador de vols que s'ha programat ha estat capaç de buscar viatges multi-destí entre diferents ciutats de la base de dades proveïda. Així, s'ha creat un mètode emprenedor en l'àmbit de la recerca de vols econòmics, amb l'avantatge de proporcionar major flexibilitat horària i augmentar les destinacions possibles. Per tant, s'ha aconseguit que l'usuari no hagi de triar un rang tancat de dates i no hagi de triar en cap moment a quina ciutat vol viatjar, sinó que aquesta esdevé la més econòmica.

S'ha pogut assolir l'objectiu principal del projecte, que era fer funcionar un programa amb les característiques necessàries per poder extreure uns resultats competitius. Tot i així, encara que s'ha atès l'objectiu principal, hi ha certs elements del programa susceptibles de millora.

La base de dades que es disposa té un volum considerable de vols. Aquests vols són únicament europeus ja que no s'ha pogut disposar d'una base de dades més àmplia. El programa desenvolupat funciona correctament amb aquest volum de dades, però caldria veure què passaria i com s'enfocaria una possible base de dades amb tots els vols mundials i si aquest algorisme ho suportaria. En cas que no es pogués realitzar amb un volum tan ampli de dades, es podria adaptar el programa per a vols domèstics d'Estats Units per exemple, ja que el volum és similar o també per una àrea amb un volum de tràfic aeri semblant.

En la base de dades amb la qual s'ha treballat, tots els vols tenen les 00:00 com a hora de sortida, ja que no es disposa de l'hora concreta. Per aquest motiu el programa no contempla escales en un mateix dia i cal separar tots els vols en franges de 1 dia per poder treballar. Una manera de perfeccionar el programa seria obtenint una base de dades amb les hores exactes de sortida de tots els vols. En aquest cas es podria classificar en franges horàries en un mateix dia i aplicar un procés semblant.

S'ha dissenyat el programa sobre una base de dades estàtica, però suposant que la base es pot actualitzar de manera periòdica. Caldria comprovar que el programa funciona correctament i mantenint el mateix format davant una base de dades dinàmica actualitzada periòdicament.

Si s'aconseguissin les millors ofertes actualitzades en tot moment, el següent pas seria aconseguir que funcionés correctament a través d'internet. És una tasca complexa ja que el llenguatge amb el que s'ha programat és diferent al que es faria servir en una pàgina web i existeix una difícil traducció entre mig.

Així doncs, a través del programa plantejat al llarg del projecte, i amb les millores aportades, es pot assolir un sistema de recerca i obtenció de bitllets d'avió econòmics capaç de competir amb la resta de cercadors i satisfer les necessitats d'una població cada cop més exigent en quant a la recerca del millor preu per a viatjar, en aquest cas, en viatges multi-destins.

Bibliografia

[1] Google, I. (13 de Septembre de 2011). *Google Flights*. Recuperat el 16 de Maig de 2015, de https://www.google.es/flights/

[2] Jose, S. (13 de Febrer de 2013). *Algoritmo de Floyd*. Obtingut de http://algoritmodefloyd.blogspot.com.es/

[3] KAYAK.com. (14 de Gener de 2004). *Kayak*. Recuperat el 16 de Maig de 2015, de Kayak: http://www.kayak.es/flights

[4] Matlab. (s.f.). *Matlab*. Obtenido de http://es.mathworks.com/

[5] Pérez Mansilla, S. (s.f.). Apunts Grafs. Universitat Politècnica de Catalunya.

[6] QL2 Software, L. (2010). *Ql2*. Recuperat el 23 de Maig de 2015, de Ql2: http://www.ql2.com/

[7] Skiplagged. (30 de Maig de 2015). *Skiplagged*. Obtingut de https://skiplagged.com/

[8] Skyscanner. (2002). *Skyscanner*. Recuperat el 16 de Maig de 2015, de Skyscanner: http://www.skyscanner.es/

[9] Vacaciones eDreams, S. (1999). *eDreams*. Recuperat el 16 de Maig de 2015, de eDreams: http://www.edreams.es/

[10] Wagner, D., Schultes, D., Sanders, P., & Delling, D. (s.f.). Engineering Route Planning Algorithms. Universität Karlsruhe (TH), 76128 Karlsruhe, Germany.

[11] *Wayna Aero S.L.* (2014). Recuperat el 23 de Maig de 2015, de https://waynabox.com/

[12] West, D. B. (1996). *Introduction to Graph Theory.*

[13] Wilson, R. J., & John J, W. (s.f.). *Graphs, An Introductory Approach.*

[14] Yan, M. (s.f.). *Dijkstra*'s Algorithm. Massachusetts Institute of Technology.

Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

ANNEXOS

TÍTOL DEL TFG: Disseny d'un algorisme d'optimització de preus en viatges multi-destí

TITULACIÓ: Grau en Enginyeria d'Aeronavegació

AUTOR: Yanik Lacroix Torrent

DIRECTOR: Susana Clara López Masip

DATA: 10 de juliol del 2015

ImportFromFile

```
function [ DataBase ] = ImportFromFile(filename, startRow, endRow,
ColumnNumber, NumericCol, StringCol)
%Import From File
%  It will Import Data From DataBase File to Matlab.

%% Initialize variables.
% filename = 'C:\Users\yanik\Desktop\DataBase.sql';
delimiter = ',';
% startRow = 3;
% endRow = 534;

%% Read columns of data as strings:
% For more information, see the TEXTSCAN documentation.
formatSpec = [repmat('%s',1,ColumnNumber) '%[^\n\r]'];

%% Open the text file.
fileID = fopen(filename,'r');


%% Read columns of data according to format string.
% This call is based on the structure of the file used to generate
this
% code. If an error occurs for a different file, try regenerating the
code
% from the Import Tool.
textscan(fileID, '%[^\n\r]', startRow-1, 'ReturnOnError', false);
dataArray = textscan(fileID, formatSpec, endRow-startRow+1,
'Delimiter', delimiter, 'ReturnOnError', false);

%% Close the text file.
fclose(fileID);

%% Convert the contents of columns containing numeric strings to
numbers.
% Replace non-numeric strings with NaN.
raw = repmat({''},length(dataArray{1}),length(dataArray)-1);
for col=1:length(dataArray)-1
    raw(1:length(dataArray{col}),col) = dataArray{col};
end
numericData = NaN(size(dataArray{1},1),size(dataArray,2));

for col = NumericCol
    % Converts strings in the input cell array to numbers. Replaced
non-numeric
    % strings with NaN.
    rawData = dataArray{col};
    for row=1:size(rawData, 1);
        % Create a regular expression to detect and remove non-numeric
prefixes and
        % suffixes.
```

```
        regexstr = '(?<prefix>.*?)(?<numbers>([-
]*(\d+[\,]*)+[\.]{0,1}\d*[eEdD]{0,1}[-+]*\d*[i]{0,1})|([-
]*(\d+[\,]*)*[\.]{1,1}\d+[eEdD]{0,1}[-+]*\d*[i]{0,1}))(?<suffix>.*)';
        try
            result = regexp(rawData{row}, regexstr, 'names');
            numbers = result.numbers;

            % Detected commas in non-thousand locations.
            invalidThousandsSeparator = false;
            if any(numbers==',');
                thousandsRegExp = '^\d+?(\,\d{3})*\.{0,1}\d*$';
                if isempty(regexp(thousandsRegExp, ',', 'once'));
                    numbers = NaN;
                    invalidThousandsSeparator = true;
                end
            end
            % Convert numeric strings to numbers.
            if ~invalidThousandsSeparator;
                numbers = textscan(strrep(numbers, ',', ''), '%f');
                numericData(row, col) = numbers{1};
                raw{row, col} = numbers{1};
            end
        catch me
        end
    end
end

%% Split data into numeric and cell columns.
rawNumericColumns = raw(:, NumericCol);
rawCellColumns = raw(:, StringCol);

%% Replace non-numeric cells with NaN
R = cellfun(@(x) ~isnumeric(x) && ~islogical(x),rawNumericColumns); %
Find non-numeric cells
rawNumericColumns(R) = {NaN}; % Replace non-numeric cells

%% Create output variable
DataBase = raw;

%% Clear temporary variables
clearvars filename delimiter startRow endRow formatSpec fileID
dataArray ans raw col numericData rawData row regexstr result numbers
invalidThousandsSeparator thousandsRegExp me rawNumericColumns
rawCellColumns R;
end
```

DepartrureCityToFirst

```
function newMatrix = DepartureCityToFirst(DailyMatrix, Departure )
% Changes the rows and columns to be the departure city the first one
%DEPARTURE IN NUMBER!

newMatrix = DailyMatrix;
```

```
newMatrix(1,:) = DailyMatrix(Departure,:);
newMatrix(Departure,:) = DailyMatrix(1,:);

DailyMatrix = newMatrix;

newMatrix(:,1) = DailyMatrix(:,Departure);
newMatrix(:,Departure) = DailyMatrix(:,1);


end
```

GetAirportName

```
function [ AirportName ] = GetAirportName( AirportNum, AirportList )
%UNTITLED6 Summary of this function goes here
%   Detailed explanation goes here
try
AirportName = AirportList{AirportNum};
catch
    disp('Wrong Airport Num or not Airport List')
end
end
```

GetAirportNum

```
function [ AirportNum ] = GetAirportNum( AirportName, AirportList )
%UNTITLED4 Summary of this function goes here
%   It gives you the number of the airport

try
AirportNum = 0;

for i=1:length(AirportList)

    if AirportList{i} == AirportName
        AirportNum = i;
    end
end

if AirportNum == 0

    fprintf('Airport with this name: %s is not in the list',
AirportName);

end

catch
    disp('Wrong Format in AirportName or Wrong AirportList')
end
```

```
end
```

GetDayNumber

```
function DayNumber = GetDayNumber( Day, Month, Year )
%UNTITLED3 Summary of this function goes he
%   Giving the month and the day, it returns the number of the day in
the
%   year. Num 1 is 1/1/2015.

try
DayNumber = Day;


for i=2015:Year
    if i == Year
            if ((fix(i/4)-(i/4)) == 0)
                YearDays = [31 29 31 30 31 30 31 31 30 31 30 31];
                DayNumber = sum(YearDays(1:Month-1))+DayNumber;

            else
                YearDays = [31 28 31 30 31 30 31 31 30 31 30 31];
                DayNumber = sum(YearDays(1:Month-1))+DayNumber;
            end


    else

            if ((fix(i/4)-(i/4)) == 0)
                 DayNumber = DayNumber + 366;

            else
                DayNumber = DayNumber + 365;
            end
    end
end
catch
    disp('Wrong Input Format of GetDayNumber Function')
    return
end
end
```

GetDayDate

```
function [ Day, Month, Year ] = GetDayDate( DayNumber )
%Given a DayNumber, the function return us the Date of this day.
%   Detailed explanation goes here
try
YearsDays = [0 365 366 365 365 365];
yearvector = [2016, 2017, 2018, 2019, 2020];
Year = 2015;

for i=2:length(YearsDays)
```

```
    if DayNumber > sum(YearsDays(1:i))
        Year = yearvector(i-1);
    end
end

YearNum = Year - 2014;
DayNumber = DayNumber - sum(YearsDays(1:YearNum));


if (Year == 2016 || Year == 2020)

    YearDays = [0 31 29 31 30 31 30 31 31 30 31 30 31];
else
    YearDays = [0 31 28 31 30 31 30 31 31 30 31 30 31];
end


for i=2:length(YearDays)
     x = sum(YearDays(1:i));
    if x < DayNumber
        Month = i;
    end
end

DayNumber = DayNumber - sum(YearDays(1:(Month)));
Day = DayNumber;
catch
   disp('Wrong Input Format of GetDayDate Function')
   return

end
end
```

MultiDestinationMatrixFlexV6

```
function [ A ] = MultiDestinationMatrixFlexV6(Phase1Days, Phase2Days,
Phase3Days, DepartureCity, DayDeparture, MonthDeparture,
YearDeparture, FromTo, NoVisit )

%Phase2Days tiene que ser un VECTOR!
%Fa que estiguem un minim de dos dies a cada ciutat, pero podem
visitar mes
%de 2 ciutats, tot depen del numero de dies que viatjem
%   Detailed explanation goes here

NumberDays = Phase1Days+sum(Phase2Days)+Phase3Days;
PriceMatrix = DepartureCityToFirst(FromTo{MonthDeparture,
DayDeparture}, DepartureCity);
FirstDay = GetDayNumber(DayDeparture, MonthDeparture, YearDeparture);

for i=(FirstDay+1):(FirstDay+NumberDays-1)
    [ Day, Month, Year ] = GetDayDate(i);
    NewPriceMatrix = DepartureCityToFirst(FromTo{Month, Day},
DepartureCity);
```

```
    if NoVisit ~= 0
    for j = 1: length(NoVisit)

        if(NoVisit(j) == 1)
            NewPriceMatrix(:,DepartureCity) = 0;
        elseif (NoVisit(j) == DepartureCity)
            NewPriceMatrix(:,1) = 0;
        else
            NewPriceMatrix(:,NoVisit(j)) = 0;
        end

    end
    end
    PriceMatrix = [PriceMatrix NewPriceMatrix];
end



%Ara hem de montar la matriu gran
[m,n] = size(FromTo{1,1});
counter = 1;


%Phase 1
Phase1Matrix = [0];

for i = 1:Phase1Days
    Matrix = [0 PriceMatrix(1,(counter+1:counter+m-1))];
    Phase1Matrix = [Phase1Matrix Matrix];
    counter = counter + m;
end


%Phase 2, Phase2Days sera ahora un vector

for p = 1:length(Phase2Days)
for i = 1:Phase2Days(p)


      Phase2{i}=[zeros(m,1) PriceMatrix(:,(counter+1:counter+m-1))];

    if i == 1
    Phase2Matrix = [Phase2{i}; zeros(m*(Phase2Days(p)-i),m)];
    else

    for k=1:i
        if k == 1
            Phase2Matrix2 = Phase2{i};
        else
            Phase2Matrix2 = [Phase2Matrix2;Phase2{i}];
        end
    end
    Phase2Matrix2 = [Phase2Matrix2; zeros(m*(Phase2Days(p)-i),m)];
    Phase2Matrix = [Phase2Matrix Phase2Matrix2];
    end

    counter = counter + m;
```

```
end
Phase2MasterMatrix{p} = Phase2Matrix;
end

largo = 0;
ancho = 0;

for i = 1:p
    [w,h] = size(Phase2MasterMatrix{i});
    if i==1
    Phase2Matrix = [Phase2MasterMatrix{i}];
    else

        Phase2Matrix = [Phase2Matrix zeros(largo,h); zeros(w,ancho)
Phase2MasterMatrix{i}];

    end
    largo = largo + w;
    ancho = ancho + h;
end


%Phase 3

for i = 1:Phase3Days
    Phase3{i}=[PriceMatrix(:,counter) zeros(m,n-1)];

    if i == 1
    Phase3Matrix = [Phase3{i}; zeros(m*(Phase3Days-i),m)];
    else

    for k=1:i
        if k == 1
            Phase3Matrix2 = Phase3{i};
        else
            Phase3Matrix2 = [Phase3Matrix2;Phase3{i}];
        end
    end
    Phase3Matrix2 = [Phase3Matrix2; zeros(m*(Phase3Days-i),m)];
    Phase3Matrix = [Phase3Matrix Phase3Matrix2];
    end

    counter = counter + m;
end



%Juntar-ho tot

[P1F,P1C] = size(Phase1Matrix);
[P2F,P2C] = size(Phase2Matrix);
[P3F,P3C] = size(Phase3Matrix);

A = [Phase1Matrix zeros(P1F,P2C+P3C); zeros(P2F,P1C) Phase2Matrix
zeros(P2F,P3C); zeros(P3F, P1C+P2C) Phase3Matrix];
[Filas Columnas] = size(A);
z = Columnas-Filas;
```

```
if z > 0
    A = [A;zeros(z,Columnas)];
end




end
```

DijkstraAlgorithm

```
function [ distance, PreD ] = DijkstraAlgorithm( A )
%UNTITLED7 Summary of this function goes here
%   From a matrix A it will perform the Dijkstra Algorithm and will
return
%   the minimum distance to reach each point and the minimum path

[Fila,Columna] = size(A);


for i=1:Fila  % EN aquest for basicament posa 9999 on hi ha 0, aixi
que no cal que es preocupem

    visited(i)=0;
    PreD(i)=0;


    for j=1:Columna

        if A(i,j) == 0

           A(i,j) = 9999;

        end

    end
end

distance = A(1,:);
distance(1)=0;
visited(1)=1;

for i=1:Fila

    min = 9999;
    %nextNodeBoole = false;

    for j=1:Fila %Busca el minim de la fila on ens trobem que encara
no estigui visitat.
```

```
        if (min > distance(j)) && (visited(j) == 0)

        min = distance(j);
        nextNode = j;
        nextNodeBoole = true;

        end

    end %Busca el minim de la fila on ens trobem


    %intentem solucionar el problemaque hi ha amb el nextNode
%     if(nextNodeBoole == false)
%         p=1;
%         find = false;
%         while p<=length(visited) && (find == false)
%             if visited(p) == 0;
%                 nextNode=p;
%                 find = true;
%             end
%             p=p+1;
%         end
%     end




    visited(nextNode) = 1;

    for c=1:Fila %Una vegada tenim el minim i sabem en quina columna
esta(nextNode)
                    %Mirem si la ruta passant pel minim es mes petita
o no
                    %que la directa que ja teniem

        if(visited(c)==0)
            if(min+A(nextNode,c) < distance(c))

                distance(c) = min+ A(nextNode,c);
            PreD(c) = nextNode;
            end
        end


    end

end

end
```

ResultatsVisualV2

```
function [Price, TravelDay, Cities2] = ResultsVisualV2( DepartureDate,
AirportList, DepartureCity, PreD, Distance,Phase1Num, Phase3Num)
```

```
%This function will allow to know visually the results of the
computation
%of the Dijstra algoeithm

%The input arguments of the function are:
%               -DepartureDay in Number
%               -Airport List
%               -Departure City in Number
%               -PreD and Distance from Dijkstra Algorithm Function

%   It returns: -Price of the Travel
%               -Travel Start Day in Date
%               -JumpDay in Date
%               -Return Day in Date
%               -Name of Departure CIty
%               -Name of City 1
%               -Name of City 2

%Ordenem la Airport List posant els aeroports en el mateix ordre que
el
%vector Distance i PreD
City1 = AirportList(DepartureCity);
AirportList(DepartureCity) = AirportList(1);
AirportList(1) = City1;

%Parametres que nesecitem
L = length(Distance);
AirportNum = length(AirportList);


for i = 1: length(AirportList)
   AirportList2(i) = i;
end


AirportNames(1) = 1;
while length(AirportNames) < L
    AirportNames = [AirportNames AirportList2];
end

% cell option with diferent options
% the vector options has the diferent airports




for j = 1:Phase3Num
    City = 0;
    City(1) = AirportNames(1);
    FirstSearch = L-(j*AirportNum)+1;
    LastAirport = PreD(FirstSearch);
    Price(j) = Distance(FirstSearch);
    TravelDay(1,j) = DepartureDate + 1 + fix (FirstSearch/AirportNum);

    i=1;
    while LastAirport ~= 0
        [m,n] = size(City);
        i = n+1;
        while i > 0
```

```
         if i > 1
             City(i) = City(i-1);
             TravelDay(i,j) = TravelDay(i-1,j);
             AirportControl = LastAirport;
         else
        City(1) = AirportNames(LastAirport);
        TravelDay(1, j) = DepartureDate + 1 +
fix(LastAirport/AirportNum);
        AirportControl = LastAirport;
        LastAirport = PreD(LastAirport);
         end
         i = i-1;
        end

    end


    if (Price(j) < 9000) && (min(Price) ~= 0)

        [m,n] = size(City);
        i = n+1;
        while i > 0
         if i > 1
             City(i) = City(i-1);

         else
             City(1) = AirportNames(1);
         end
         i=i-1;
        end


    if j == 1

        Cities = City';

    else

        if length(Cities) ~= length(City)

            if length(Cities) > length(City)
                City = [City 0];
            else
                [w,h] = size(Cities);
                Cities = [Cities; zeros(1,h)];
            end

        end




        Cities = [Cities City'];
    end
        else
        Price(j) = 0; % si el preu es 0 vol dir que hi ha hagut un
error
    end
```

```
end

if exist('Cities','var') == 1
[m,n] = size(Cities);
for i=1:m
    for j=1:n
        if Cities(i,j) ~= 0
        Cities2{i,j} = char(AirportList(Cities(i,j)));
        else
           Cities2{i,j} = [];
        end
    end
end
else
    Cities2 = 0;
end
end
```

MainIterative

```
SetDay ='Set Start of Disponibility\n';
DateDep = input(SetDay,'s');
StartDay =
GetDayNumber(str2double(DateDep(1:2)),str2double(DateDep(4:5)),str2dou
ble(DateDep(7:10)));
%StartDay = 32;

SetEndDay = 'Set End Day of Disponibility \n';
DateEnd = input(SetEndDay,'s');
EndDay =
GetDayNumber(str2double(DateEnd(1:2)),str2double(DateEnd(4:5)),str2dou
ble(DateEnd(7:10)));

%EndDay = 40;

SetDepartureCity = 'From which city do you want to start your travel?
\n';
DCity = input(SetDepartureCity,'s');
DepartureCity = GetAirportNum(DCity, AirportList);
%DepartureCity = 3;

SetMinDays = 'Minimum Traveling Days \n';
MinDays = input(SetMinDays);

SetMaxDays = 'Maximum Traveling Days \n';
MaxDays = input(SetMaxDays);

Phase1Days = (MaxDays-MinDays);
Phase2DaysNum = MaxDays - ((MaxDays-MinDays)*2);
Phase3Days = (MaxDays-MinDays);


%Set cities that you don't want to visit
```

```
NoVisit = [7]; %s'han de posar en un vector amb les ciutats en numero,
si no en volem cap, 0.
Eliminar = 0;
for i = StartDay:EndDay
%Parameters
Phase2Days = 0;
DepartureDate = i;
[DepartureDay, DepartureMonth, DepartureYear] =
GetDayDate(DepartureDate);

x = 1;
j = 0;
while Phase2DaysNum/x >= 2
    a = Phase2DaysNum/x;
    for y = 1:x
        Phase2Days(y) = fix(a);
    end
    len = length(Phase2Days);

    totalok = false;
    num = 1;
    middle = ceil(len/2);
    while totalok == false
        if sum(Phase2Days) ~= Phase2DaysNum
            if num == 1
            Phase2Days(middle) = Phase2Days(middle) + 1;
            elseif fix(num/2) == num/2
               Phase2Days(middle-fix(num/2)) = Phase2Days(middle-
fix(num/2)) + 1;
            else
                Phase2Days(middle+fix(num/2)) =
Phase2Days(middle+fix(num/2)) + 1;
            end

            num = num +1;
        else
            totalok = true;
        end
    end
    totalok = false;



%Functions
A = MultiDestinationMatrixFlexV6(Phase1Days, Phase2Days, Phase3Days,
DepartureCity, DepartureDay, DepartureMonth,DepartureYear, FromTo,
NoVisit); %Departure City, DepDay, DepMonth, DepYear, FromTo
[ distance, PreD ] = DijkstraAlgorithm( A );
%Funció comprovant Dijkstra
%Seguir fent la funció perque la matriu s'ajusti al numero de dies
%Mirar base de dades per agafar mes dades i de millor manera
[Price, TravelDay, Cities] = ResultsVisualV2( DepartureDate,
AirportList, DepartureCity, PreD, distance, Phase1Days, Phase3Days );

    if(min(Price) > 0)
if i+j == StartDay
    MasterPrice = Price;
    MasterTravelDay = TravelDay;
    MasterCities = Cities;
else
```

```
    MasterPrice = [MasterPrice Price];

    [f1,c1] = size(MasterTravelDay);
    [f2,c2] = size(TravelDay);

    if f1 > f2
        TravelDay = [TravelDay; zeros(f1-f2,c2)];
        Cities = [Cities; cell(f1-f2,c2)];

    elseif f2 > f1
        MasterTravelDay = [MasterTravelDay; zeros(f2-f1,c1)];
        MasterCities = [MasterCities; cell(f2-f1,c1)];
    end
    MasterTravelDay = [MasterTravelDay TravelDay];
    MasterCities = [MasterCities Cities];

end
end

x = x+1;
j = j+1;
end




end


[m n] = size(MasterCities);
count = 1;
Found = 0;
%if max(Eliminar) < n
for x = 1:n
    Found = 0;
       for y = 2:m

           for z = 2:m
               if y ~= z
                   if Found == 0
                       if isempty(MasterCities{y,x}) ||
isempty(MasterCities{z,x})
                       else
                if MasterCities{y,x} == MasterCities{z,x}
                    Eliminar(count) = x;
                    count = count + 1;
                    Found = 1;
                end
                       end
                   end
               end
           end
       end

end
if Eliminar ~= 0
```

```
MasterCities(:,Eliminar) = [];
MasterTravelDay(:,Eliminar) = [];
MasterPrice(:,Eliminar) = [];
end
%end
ExportData(MasterCities, MasterPrice, MasterTravelDay)
```

ExportData

```
function ExportData(MasterCities, MasterPrice, MasterTravelDays)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here

[m,n] = size(MasterTravelDays);

for i = 1:m
    for j = 1:n

        if MasterTravelDays(i,j) ~= 0
        [ Day, Month, Year ] = GetDayDate(MasterTravelDays(i,j));
        barra = num2str('/');
        MasterTravelDays2{i,j} = [num2str(Day) barra num2str(Month)
barra num2str(Year)];
        end
    end
end

for i = 1:length(MasterPrice)
    MasterPrice2{1,i} = MasterPrice(i);
end

Master = [MasterPrice2;MasterCities;MasterTravelDays2];
xlswrite('C:\Users\yanik\Desktop\Optimització\Prova',Master,'Hoja1','B
2');


end
```